NPS-52-87-035

NAVAL POSTGRADUATE SCHOOL

Monterey, California

AN INTERACTIVE COMPUTER GRAPHICS NETWORK

MONITOR FOR A TACTICAL COMMUNICATIONS NETWORK

Laurence W. Griggs

Richard A. Adams

Michael J. Zyda

August 1987

Prepared for:

Naval Ocean Systems Center
San Diego, CA 92152

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. C. Austin
Superintendent

D. A. Schrady
Provost

This work was supported by a grant from the Naval Ocean Systems Center, San Diego (Ref. # N000148WX4B418AB). This work was generated from Laurence W. Griggs' Masters Thesis.

This report was prepared by:

UCLASSIFIED
CRITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

EPORT SECURITY CLASSIFICATION UCLASSIFIED	1b RESTRICTIVE MARKINGS
ECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release Distribution unlimited
ECLASSIFICATION / DOWNGRADING SCHEDULE	
RFORMING ORGANIZATION REPORT NUMBER(S) NS52-87-035	5. MONITORING ORGANIZATION REPORT NUMBER(S)
NAME OF PERFORMING ORGANIZATION val Postgraduate School — 6b OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Naval Ocean Systems Center
ADDRESS (City, State, and ZIP Code) nterey,CA 93943	7b. ADDRESS (City, State, and ZIP Code) San Diego, CA
NAME OF FUNDING / SPONSORING ORGANIZATION aval Ocean Systems Center — 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N000148WX4B418AB
ADDRESS (City, State, and ZIP Code) an Diego, CA 92152	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

TITLE (Include Security Classification)

Interactive Computer Graphics Network Monitor for a Tactical Communications Network

PERSONAL AUTHOR(S)

urence W. Griggs, Richard A. Adams and Michael J. Zyda

TYPE OF REPORT	13b TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
chnical	FROM _____ TO _____	August 1987	87

SUPPLEMENTARY NOTATION

COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

ABSTRACT (Continue on reverse if necessary and identify by block number)

In order to apply modern networking technology, either circuit- or packet-switched, to ictical communication networks, network designers must develop (1) robust link level proto-ols to handle broadcast media and node mobility and (2) distributed, adaptive routing pro-ocols to handle the rapid reconfigurations required by node mobility and mortality. In ldition, from the network manager's point of view, combat imposes electronic order of battle onstraints that can affect network performance and limit the available network configur-ions. Optimizing message throughput under such design and operational constraints is xtremely difficult.

Intended as an aid to both network designers and managers, this study describes a network onitor that uses modern high-speed graphics hardware and a responsive multi-window user nterface to depict, in real-time, the state of a packet- or circuit-switched tactical com-unications network. We model the network state as a set of overlays to an existing well-nown tactical display format, namely that of Naval Tactical Data System (NTDS). In our

0. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
2a. NAME OF RESPONSIBLE INDIVIDUAL ichael J. Zyda	22b. TELEPHONE (Include Area Code) 408/646-2305 — 22c. OFFICE SYMBOL

D FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

☆ U.S. Government Printing Office: 1986—606-24.

implementation, tactical and network displays are decoupled in order to allow the network monitor's use with other graphical combat information systems.

An Interactive Computer Graphics Network Monitor for a Tactical Communications Network ‡

Laurence W. Griggs, Richard A. Adams and Michael J. Zyda *

Naval Postgraduate School,
Code 52, Dept. of Computer Science,
Monterey, California 93943

ABSTRACT

In order to apply modern networking technology, either circuit- or packet-switched, to tactical communications networks, network designers must develop (1) robust link level protocols to handle broadcast media and node mobility and (2) distributed, adaptive routing protocols to handle the rapid reconfigurations required by node mobility and mortality. In addition, from the network manager's point of view, combat imposes electronic order of battle constraints that can affect network performance and limit the available network configurations. Optimizing message throughput under such design and operational constraints is extremely difficult.

Intended as an aid to both network designers and managers, this study describes a network monitor that uses modern high-speed graphics hardware and a responsive multi-window user interface to depict, in real-time, the state of a packet- or circuit-switched tactical communications network. We model the network state as a set of overlays to an existing well-known tactical display format, namely that of Naval Tactical Data System (NTDS). In our implementation, tactical and network displays are decoupled in order to allow the network monitor's use with other graphical combat information systems.

‡ This work was supported by a grant from the Naval Ocean Systems Center, San Diego (Ref. # N0001487WX4B418AB). This work was generated from Laurence W. Griggs' Masters Thesis.

* Contact author.

TABLE OF CONTENTS

I. INTRODUCTION 6

II. BACKGROUND 10

A. UNT NETWORK ARCHITECTURE 10

B. ATD/UNT NETWORK ARCHITECTURE 11

C. REQUIREMENTS FOR THE ATD/UNT NETWORK MONITOR 12

III. GRAPHICAL DEPICTION OF THE ATD/UNT NETWORK 17

A. CHARACTERISTICS OF THE ATD/UNT NETWORK 17

B. DISPLAYING NETWORK CHARACTERISTICS 18

C. A BACKGROUND PICTURE FOR THE NETWORK DISPLAY 18

D. THE UNETGRAF DISPLAYS 19

E. NETWORK-ONLY MODE 20

IV. FUNCTIONAL SPECIFICATIONS FOR THE UNETGRAF SYSTEM 24

A. IDENTIFICATION OF EXTERNAL SYSTEMS 24

1. The User 24

2. The Network Administrator 25

3. The Naval Tactical Data System (NTDS) 27

B. COMMUNICATION WITH EXTERNAL SYSTEMS 27

1. Messages from the User to UNETGRAF 28

2. Messages between the Network Administrator and UNETGRAF 28

3. Messages between the NTDS Translator and UNETGRAF 29

V. UNETGRAF ARCHITECTURAL DESIGN 31

A. THE UNETGRAF.C MODULE 31

B. THE USER INTERFACE SUBSYSTEM 32

C. THE OBJECTS.C MODULE 32

D. THE DRAW_NTDS.C MODULE 33

E. THE DRAW_DETAIL.C MODULE 33

F. THE CORRELATE.C MODULE 33

G. THE TACTICAL DISPLAY STORAGE SUBSYSTEM 33

H. THE NETWORK DISPLAY STORAGE SUBSYSTEM 34

I. THE UTILITIES SUBSYSTEM 35

VI. SUMMARY AND CONCLUSIONS 42

APPENDIX A - THE UNETGRAF USER'S GUIDE 45

APPENDIX B - THE MODIFIED IRIS WINDOW MANAGER 64

LIST OF REFERENCES 85

INITIAL DISTRIBUTION LIST 87

I. INTRODUCTION

Commanders have the doctrinal obligation to maintain communications with forces they send in harms's way, but have usually been forced by the state of communications technology to rely on a few tenuous, easily jammed, low-capacity radio links to coordinate and direct any required supporting arms and reserve formations. Now, networking techniques, both circuit-switched and packet-switched, promise to increase the number of links available and to make these links more robust.

Tactical communications, however, present a set of difficult challenges for the network designer:

- Nodes are subject to hostile disruption (jamming, for example) and/or destruction.
- Most nodes are mobile.
- Broadcast media (i.e., radio communications) are generally required.

These factors dictate that the network designer provide each node with the ability to make valid independent decisions about routing packets or circuits to the destination node. In networking parlance, the routing protocol is said to be distributed and adaptive [STAL85].

The validity of decisions about routing and the validity of other decisions in the areas of media contention, error control and flow control is ultimately decided

by message throughput. But network management--the science of maximizing throughput--is possible only if some measures of network performance are captured. Those portions of the network system that the designer has included to obtain such measurements are collectively known as the network monitor. Figure 1.1, a generic system-level diagram of a centralized network monitor, shows "Interactive Displays" as one the monitor's subsystems [TERP82].

The objectives of this study are

- determine the appropriate interactive computer graphics displays for the network monitor of a prototype tactical communications network presently under development by the Naval Ocean Systems Center (NOSC) and

- develop a user interface to support the use of these displays.

To accomplish these objectives, it has been necessary to develop a small-scale software system (8000 lines of C source code) which NOSC has indicated will be ported from its development environment, an IRIS graphics workstation, to a Sun workstation [GRIG87]. Thus, the form of this study is that of an analysis and design document intended for (1) personnel assigned the tasks of porting, improving or otherwise maintaining this specific software system and (2) future designers of tactical communications network monitors who may find this particular design instructive.

The software life cycle model is the best understood and most familiar means of describing software development issues and it is used as a framework for this study. Since there is some lack of agreement on what phases actually constitute

the life cycle, the definitions found in [BERZ86] will be used; to wit:

Requirements definition:
: define purpose of the proposed software system; establish constraints on the design process.

Functional specification:
: define a user model of the system that satisfies the requirements; include only those aspects of the system relevant to the user.

Architectural design:
: decompose the system into a set of modules; each module being defined by a set of black box specifications.

Module design:
: define algorithms and data structures used to implement each black box specification.

Implementation:
: implement each module in a programming language.

Testing:
: perform unit, system and end-user testing.

Evolution:
: add new functionalities to a delivered system.

Repair:
: repair faults that are discovered after system delivery.

Chapter II provides background on NOSC's tactical communications network, known as the Unified Networking Technology (UNT) network. Also in Chapter II, we present the requirements for the UNT network's Interactive Display System. Chapter III discusses an appropriate set of displays to depict the operation of the UNT network. Chapter IV presents the system's functional specifications. Chapter V describes the modular decomposition of the system. Chapter VI summarizes the study and looks to the future of tactical communications network monitoring and control systems.

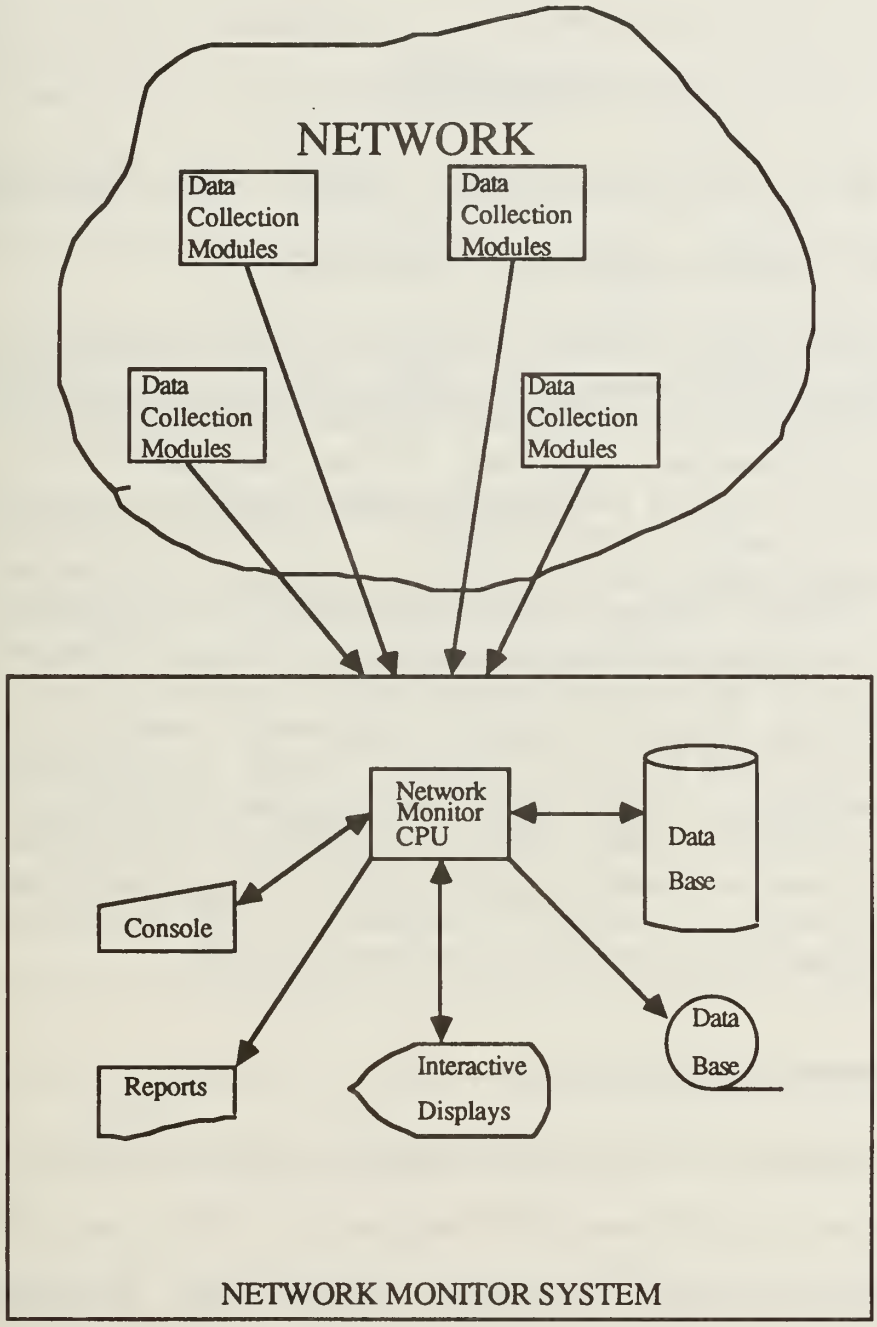

Figure 1.1 Generalized Network Monitor System

II. BACKGROUND

The goal of the Naval Ocean Systems Center's Unified Networking Technology (UNT) project is summarized in the following quote:

> Naval platforms need to be able to interrogate a variety of remote data sources, combine the data obtained with decision aids to produce tailored products in support of various warfare scenarios, and reliably distribute the product to the supporting commanders. Existing and planned Naval combat systems do not have standardized interface protocols that permit inter-system data transfer. Computer systems . . . do not have a standard set of [communication] protocols. This results in unique and costly software development. Similarly, existing Naval telecommunications is accomplished by means of a large number of individual networks and links, each designed for a specific service and operating independently of the others. There is little capability for automatic transfer of information between these networks and links, and they are vulnerable to enemy countermeasures and/or are unreliable. A Unified Networking Technology Architecture is being studied in order to provide reliable automated network operation and internetwork information transfer to support evolving Fleet command and control requirements and provide for expansion of the battle area, compressed reaction times and high volumes of data [SPRA86].

A. UNT NETWORK ARCHITECTURE

Figure 2.1 depicts a high-level view of the system envisioned for a communications node in the full-scale UNT "multi-network". The major software modules of this UNT design are the Link Controller (LC), Multinetwork Controller (MC) and the Network Administrator (NA). The positions of these modules in the software layers underlying the UNT network architecture are

shown in Figure 2.2. Note the correspondence of the MC and LC modules/layers to their well-known counterparts in the International Standards Organization's (ISO) Open System Interconnection (OSI) model. This is an important design feature of UNT which promises to provide a significant commonality with mainstream research on communications networking. The NA module performs no actual networking functions itself but provides the storage and retrieval functions for the other UNT components, including the network monitor.

B. ATD/UNT NETWORK ARCHITECTURE

The Advanced Technology Demonstration of the UNT network (hereinafter referred to as ATD/UNT) is scheduled for mid-FY1990. Its goal is to demonstrate that the UNT network can support the full-range of intra-battle group communications consisting of voice, tactical data and normal TTY message traffic [CASE86]. Specific objectives are

- guaranteed TTY message delivery within two minutes of transmission,
- network reconfigurations completed within 30 seconds of node failure,
- extension of battle group communications range to 1000 nautical mile radius, and
- effective use of idle capacity (i.e., demonstration of a workable adaptive routing protocol).

Figure 2.3 depicts the planned system architecture of an ATD/UNT node. On the physical level, each node will be equipped with up to three communication channels, implemented as one single-channel UHF radio and two single-channel

HF radios. All transmissions will be digitized by a 2400-baud modem on each channel. The Link Controller (LC) protocols (media access, forward error control and flow control) for the UHF channel will probably be different from those for the HF channel. Forward error control will not be required for voice and tactical data transmissions due to their "perishable" nature. The ATD/UNT network will consist of five or six such nodes in various combinations of land-, ship- and aircraft-based configurations. The software architecture will be identical to that of the full-scale UNT node but only intra-battle group communications will be supported [CASE86].

C. REQUIREMENTS FOR THE ATD/UNT NETWORK MONITOR

Since the UNT project is just beginning and the ATD demonstration will not be conducted for nearly three years, only very general requirements have been established for the ATD/UNT Interactive Display system that we have been tasked to build. These requirements are summarized below:

1. System to be built

An interactive, computer-graphics-based network monitor for the Advanced Technology Demonstration of the Unified Networking Technology Program. Short title: UNETGRAF.

2. Requirements:

- Requirement 1. Examine the computer graphics animation capabilites required to monitor the flow of information through a Multinetwork Controller. Prototype displays will be constructed in a development environment

consisting of the C programming language, UNIX operating system and Silicon Graphics IRIS workstation [ZYDA87].

- Requirement 2. Develop a prototype user interface to support the use of the developed computer graphics animations. This user interface will be built around the IRIS Window Manager [ZYDA87].

- Requirement 3. Support demonstrations of the developed software in conjunction with demonstrations of NOSC's Multinetwork Controller [ZYDA87].

- Requirement 4. Make maximum use of the standard Naval Tactical Data System (NTDS) symbol set where possible [GRIG87].

- Requirement 5. Since the host platforms for the ATD/UNT nodes will be ships and aircraft within the naval battle group, make provisions to use the positioning data available from the host's NTDS system to produce as realistic a network depiction as possible. However, ensure that a subset implementation (i.e., without this positioning data) will still adequately depict network operation [GRIG87].

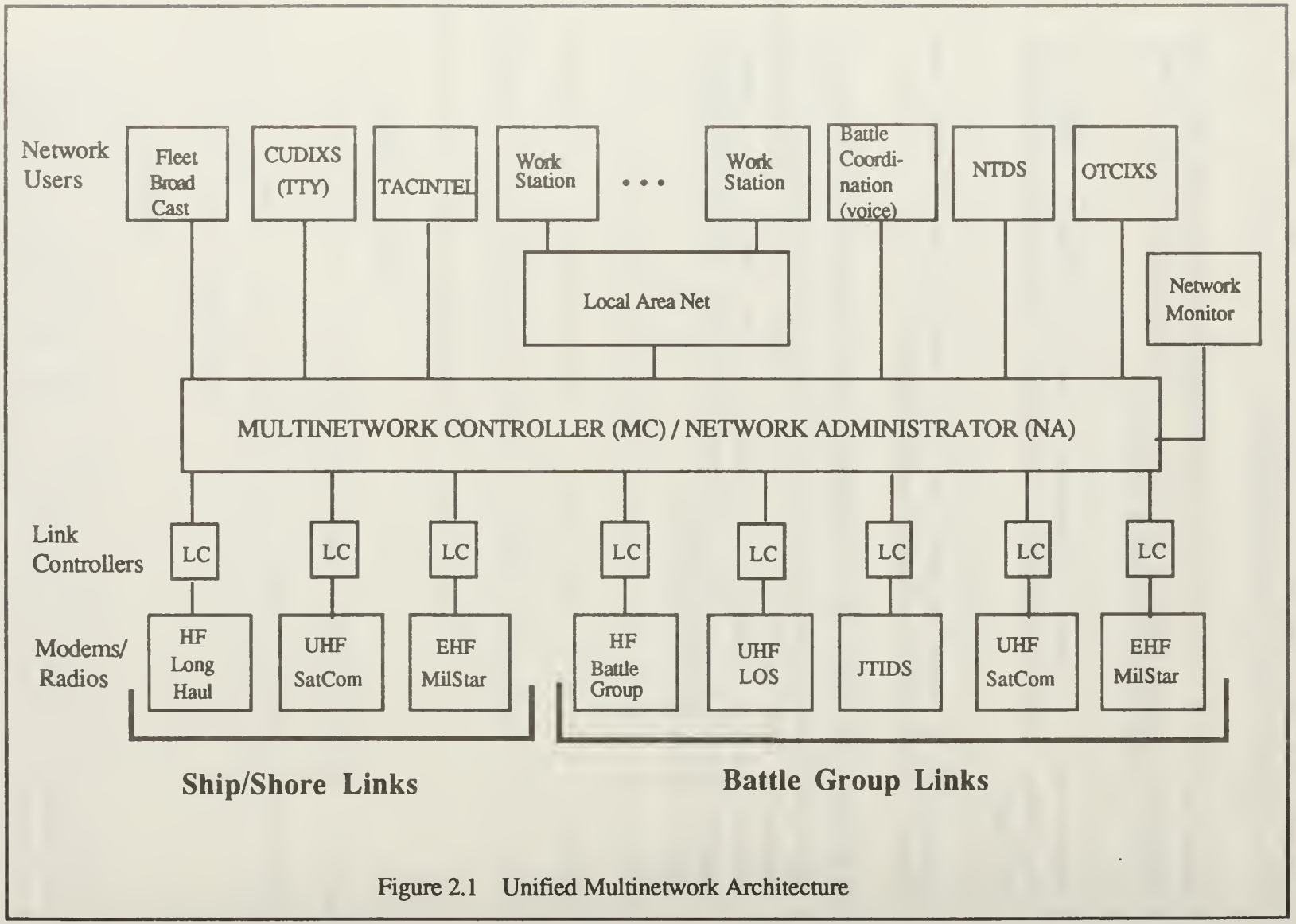

Figure 2.1 Unified Multinetwork Architecture

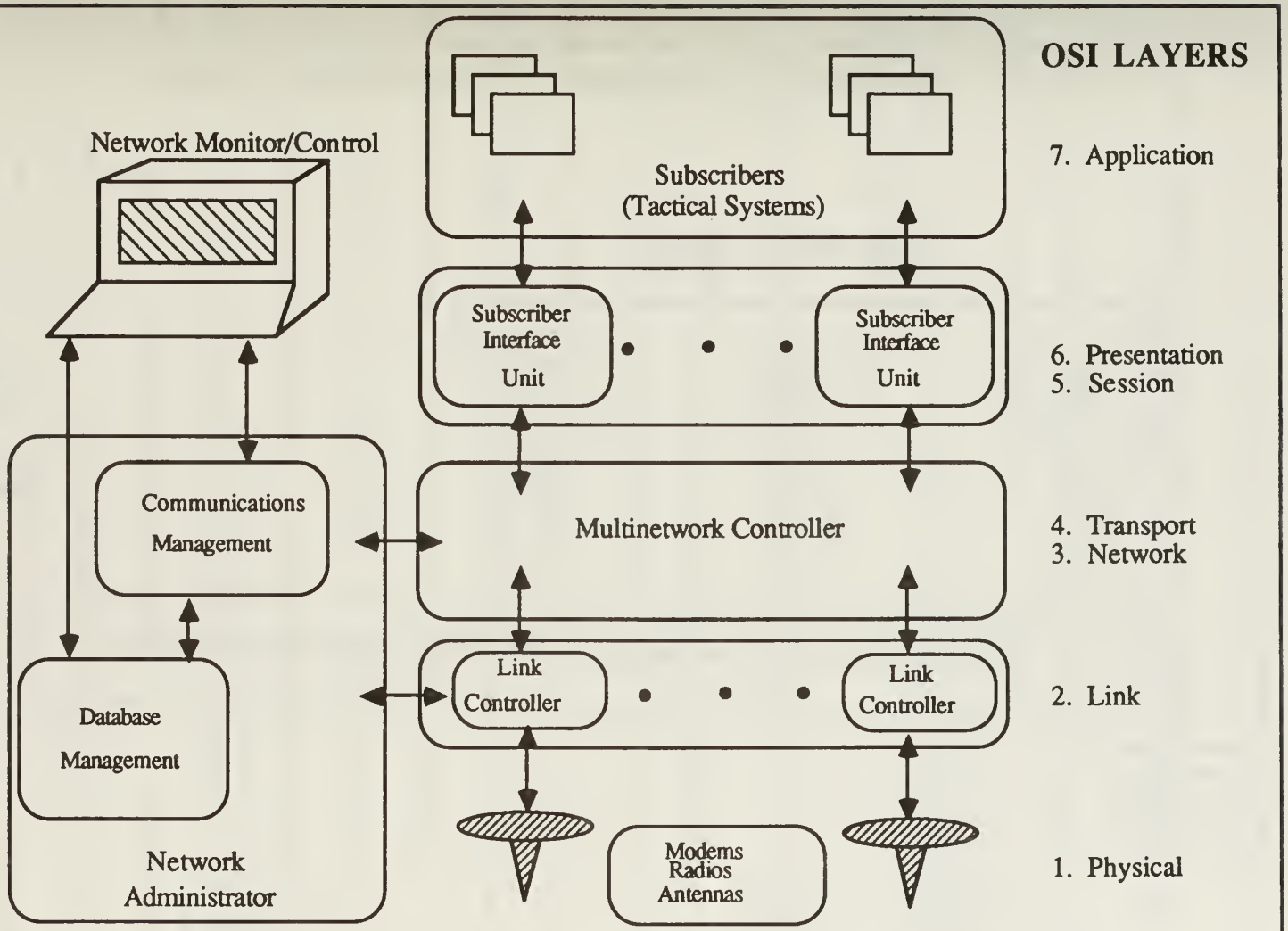

Figure 2.2 UNT Network Software Modules and Layers

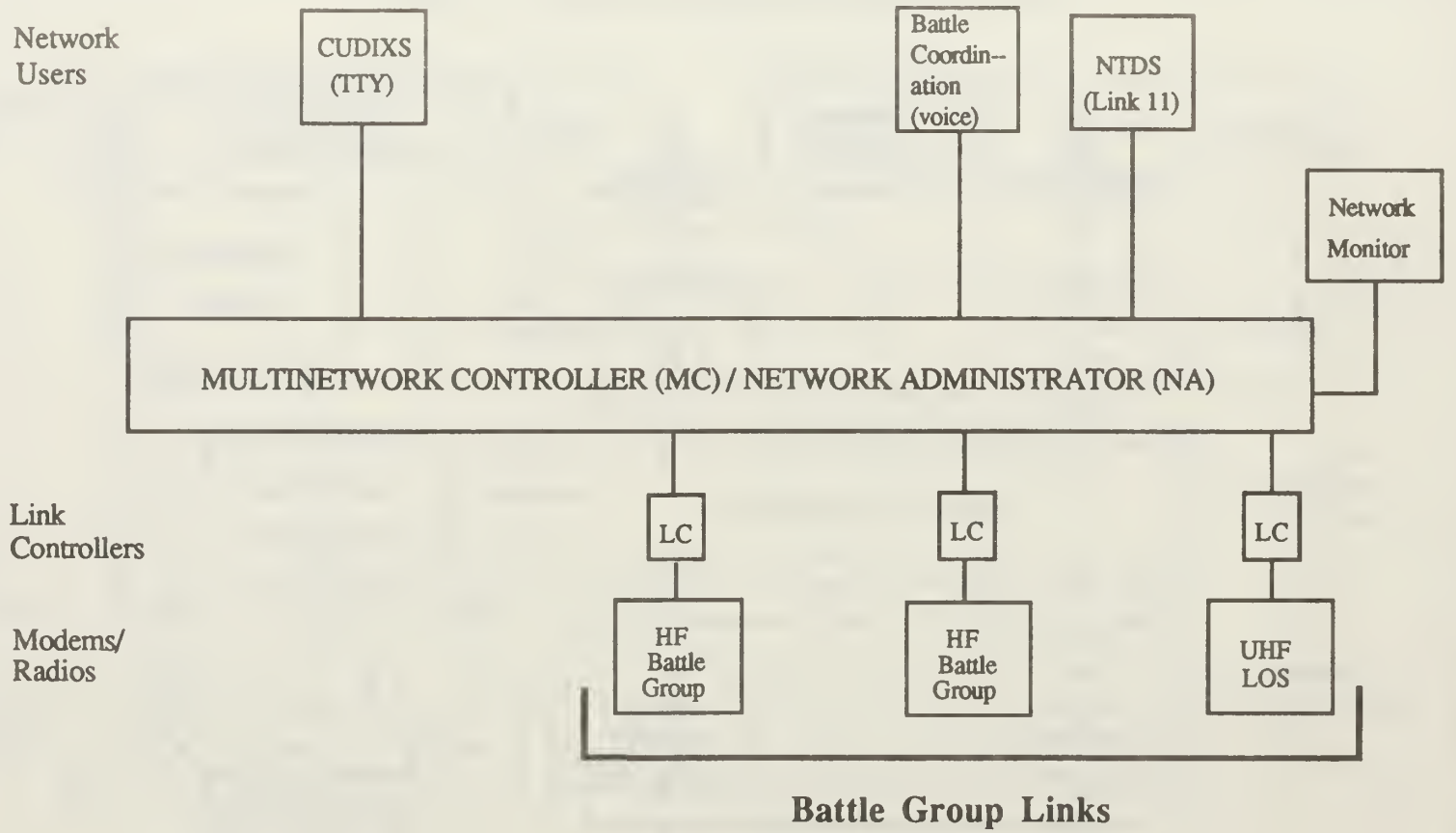

Figure 2.3 Advanced Technology Demonstration (ATD) Architecture

III. GRAPHICAL DEPICTION OF THE ATD/UNT NETWORK

The previous chapter described the UNT network and stated the general requirements for the UNETGRAF system. In this chapter, we intend to refine these requirements into a set of displays that we believe adequately depict the state of the ATD/UNT network.

A. CHARACTERISTICS OF THE ATD/UNT NETWORK

Over the course of several months during early CY1990, ship and aircraft availability permitting, the ATD/UNT network will evolve from a land-based configuration to a fully mobile configuration with its nodes hosted by ships and aircraft [CASE86]. This final fully mobile configuration is the true test of the UNT concept and we assume this configuration throughout this study.

The instantiation of the MC/LC/NA at each node makes clear the ATD/UNT network's distributed nature. In this decentralized situation, it will be the task of these three subsystems to infer as much as possible about the true network topology from the limited information they receive. These inferences will then be combined with any available information on local and remote node performance parameters to adaptively route messages, either self-originated or relayed, to their respective destination nodes.

B. DISPLAYING NETWORK CHARACTERISTICS

No matter how difficult it is to obtain, a network's topology is still representable by a directed graph. Similarly, a routing protocol may well involve sophisticated algorithms, but its result can still be depicted as a trace between source and destination vertices on the same directed graph. The proper degree of abstraction for a graphics-based network monitor is at this level, where underlying complexities are hidden and only external behavior is displayed. Node and link performance must, of course, be monitored, but depiction of such lower level details is usually desired only when exceptional conditions occur. The identification of these exceptional conditions by a visual or aural alarm is a feature of most existing network monitors [TERP82]. It is assumed that after recognition of the alarm condition, the user will investigate these lower levels of detail to determine its possible cause.

C. A BACKGROUND PICTURE FOR THE NETWORK DISPLAY

Figure 3.1 shows a directed graph as an abstraction of the ATD/UNT network. A dotted line depicts a routing trace between Node 1 and Node 4. Though a good bit of information about the network is provided by this abstraction, much more cognitive content is imparted by Figure 3.2, where the same topology and route is depicted but in which the vertices are not merely triangles but actual ships whose network connectivity is affected by such factors

as weather, distance, hostile jamming and terrain masking, none of which can be ascertained from Figure 3.1.

The background picture in Figure 3.2 is an NTDS tactical display which is itself an abstraction, albeit a familiar one to naval tacticians. We believe its inclusion as an integral part of the UNETGRAF display provides elements of realism and familiarity that will enable a much wider audience to understand the objectives of the ATD/UNT network and appreciate the complexities involved in their attainment.

D. THE UNETGRAF DISPLAYS

The foregoing analysis leads us to offer five different but related displays as those most appropriate for the UNETGRAF system. A sample of each display is provided in Appendix A (UNETGRAF User's Guide).

The *TacticalDisplay* consists of a near-standard NTDS display based on an original implementation for the IRIS workstation by Adams [ADAM87]. Responsive field-of-view controls are provided to enable rapid movement to any portion of the display that might be of interest to the user.

The *NetworkOverlay* consists of node symbols (triangles) superimposed on the NTDS symbols of their respective hosts. The *TacticalDisplay* and the *NetworkOverlay* are always visible and are not subject to any user controls other than those involving field-of-view adjustments.

The *ConnectivityOverlay* appears as a directed graph with vertices at any node selected by the user. The presence of a solid line indicates an adjacency (one hop) relationship exists between the nodes at the line's end points. (Though all adjacencies are bi-directional, we use an arrowhead to denote which end-point's adjacencies are presently displayed.)

The *RoutingOverlay* consists of one node designated "Source" by the user and an arbitrary number of nodes designated as "Destination". The routes between source and destinations are depicted by a set of dotted directional traces.

The *NodeDetailDisplay* depicts the performance of an ATD/UNT node over a user-specified time interval. Performance parameters of interest are packet counts in several categories: voice, NTDS data, TTY and network management. (We have assumed that the ATD/UNT network will be packet- versus circuit-switched.) Error statistics and idle capacity are also depicted [GRIG87]. The *NodeDetailDisplay* represents a step down in detail and is capable of being either visible or hidden, depending on the user's desires.

E. NETWORK-ONLY MODE

Requirement (5) for the UNETGRAF system specified that the system must provide a useful picture of network operation and performance **without** NTDS positioning data. We have met this requirement by modeling the communication nodes as either "correlated" or "uncorrelated". A node is "correlated" if its host can be located in the (possibly empty) set of NTDS contacts for which positioning

data is available; if the host is not located, then the node is "uncorrelated". Uncorrelated nodes are displayed in default positions relative to the NTDS data link reference point (DLRP).. If no NTDS positioning data is available, all nodes are uncorrelated. Figure A.6 shows such a "network-only" display.

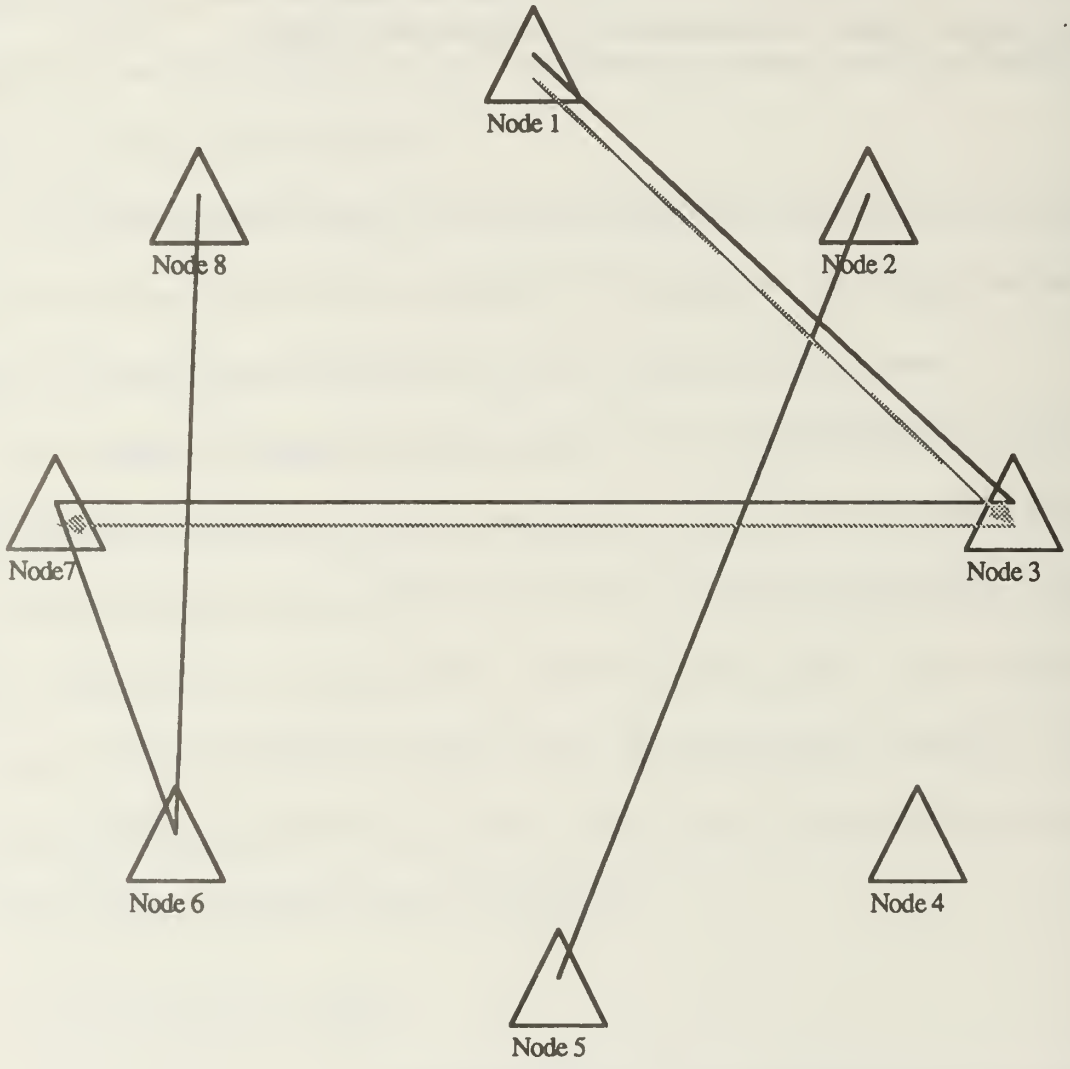

Figure 3.1 An Abstract Connectivity and Routing Diagram

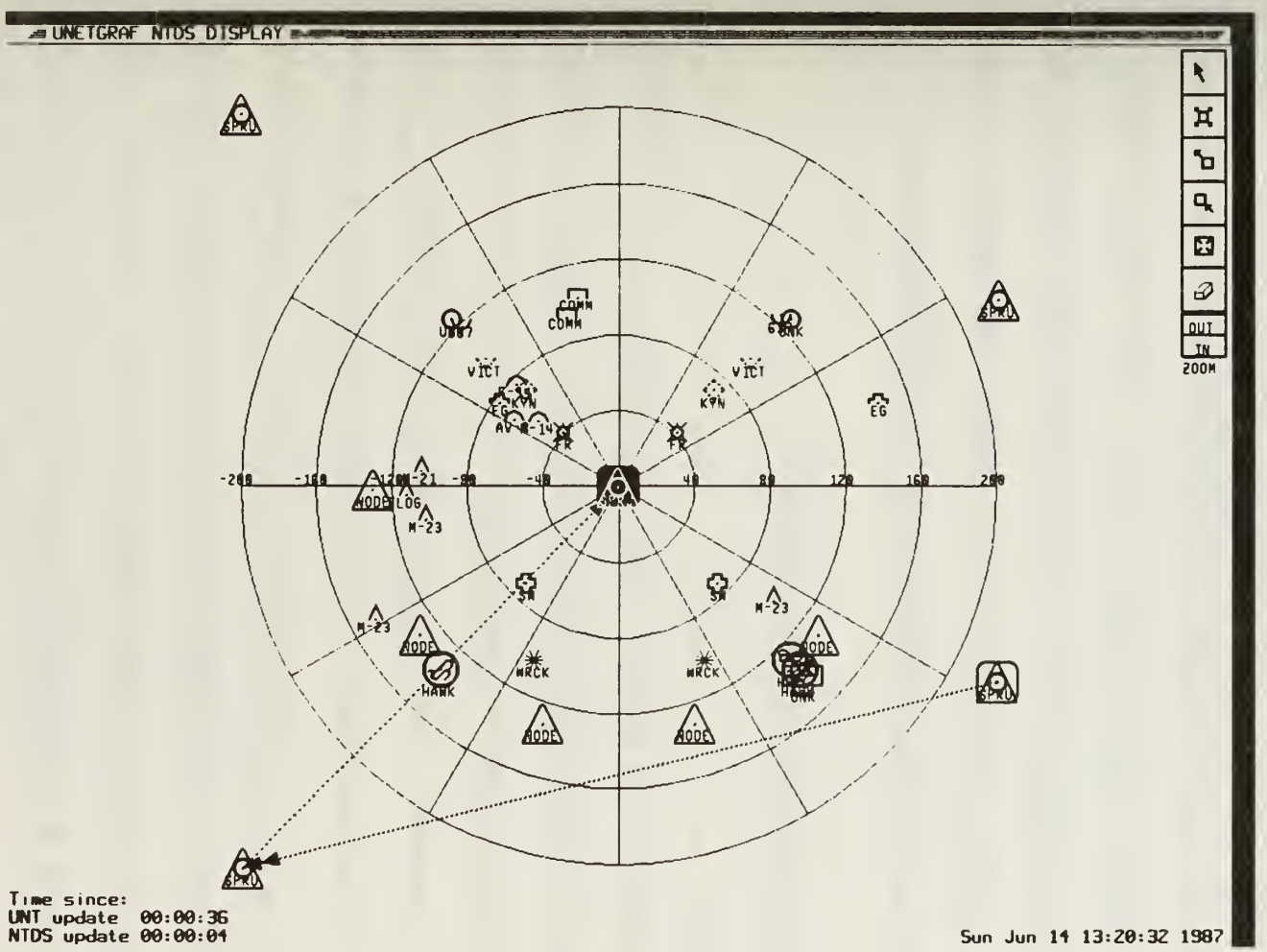

Figure 3.2 The UNETGRAF Routing Overlay

IV. FUNCTIONAL SPECIFICATIONS FOR THE UNETGRAF SYSTEM

In the previous chapter, it was established that our depiction of the ATD/UNT network was to be based on an NTDS display with user-selectable connectivity overlays, routing overlays and node detail displays. Clearly, UNETGRAF must communicate with external systems in order to produce such displays. The objective of this chapter is to determine the identity of these external systems and to define their interfaces with the UNETGRAF system.

A. IDENTIFICATION OF EXTERNAL SYSTEMS

1. The User

UNETGRAF's first external "system" is its user. As an interactive computer graphics system, UNETGRAF has an implicit requirement to provide the user with rapid and consistent response to his input. The exact nature of this input must be considered: should it consist only of pointing/clicking with the mouse or should text input also be accepted? Since UNETGRAF is a monitor versus a control device, the user's actions can be limited to (1) altering the field of view by manipulation of the underlying tactical display, (2) obtaining amplifying textual information about an NTDS contact, (3) selection of the nodes, if any, that are to appear in the connectivity and routing overlays, (4) calling up, modifying, or closing node detail displays for individual nodes, and (5) disabling

various warnings and alarms that appear on the display. Since all these interactions can be performed by a pointing device, we have made no provisions within UNETGRAF to recognize any form of input other than that provided by the mouse.

The number of UNETGRAF users is also an issue under this heading. We anticipate that at some ATD/UNT nodes it may be necessary for several users to monitor the network's operation. In such a situation we could expect that multiple display terminals, each capable of independent user input, would be installed. A single UNETGRAF process should be capable of serving users at each of these terminals.

The number of possible user actions described above can be quite large--numbering literally in the hundreds. Additionally, there may be multiple users. Based on these circumstances, we have developed an entire subsystem solely to handle user interaction. A central feature of this subsystem is the use of a data structure, of a type hereinafter called *user_struct,* to record the state of each user's interaction with the UNETGRAF system.

2. The Network Administrator

The second external system we must consider is the ATD/UNT Network Administrator (NA) which contains the database used by the Multinetwork Controller, Link Controller, and UNETGRAF. Since the NA is not yet designed, we have had to make a number of assumptions about its content and function.

The information it collects will have to be obtained from (1) the activities of its local node, (2) information received from adjacent nodes and (3) inferences based on network traffic analysis. Due to the ATD/UNT network's decentralized architecture, the NA cannot be expected to be omniscient, but its goal is to make available to its users as much information about the complete network topology as it possibly can. As a user of the NA, UNETGRAF can expect to obtain at least partial replies to the following messages: (1) which nodes are members of the network, (2) how these nodes are connected--who is talking to whom, (3) how network traffic is being routed from the local node to all possible destination nodes and (4) how the network is performing in terms of message throughput and link utilization.

Though the ATD/UNT network is in the class of networks that are described as "rapidly-reconfigurable", the replies provided by the NA to these messages will remain current for some period of time. We assume it will be more efficient during the period between "UNT updates" for UNETGRAF to hold a local copy of the replies than to repeatedly send messages to the NA, possibly over a local area network. This local copy of the network's state consists of (1) a *Nodelist* containing a sequence of records called *node_reports,* (2) a *ConnectivityMatrix* containing all known adjacency relations that exist between nodes, and (3) a *RoutingTable* containing a description of the path to each destination node that this local node considers to be the optimal route.

3. The Naval Tactical Data System (NTDS)

Though many aspects of the NTDS system are classified, we can state here, in general terms, that NTDS permits the ships and aircraft in the naval battle group to periodically share information on the tactical situation, including the current position of all friendly ships and aircraft. Some of these friendlies will be hosts to the ATD/UNT nodes and should be able to share this positioning information with UNETGRAF. We have made the following assumptions about the UNETGRAF/NTDS interface:

- A software translator layer entitled "NTDS Translator" will be necessary. This translator will have the two major tasks of (a) translating NTDS bit streams into an ASCII-format record usable by UNETGRAF. (hereafter this record will be referred to as a *contact_report)* and (b) appending the node's network address to the *contact_report* of any NTDS contact which is a host for an ATD/UNT node.

- UNETGRAF maintains an internal store of *contact_reports* called a *ContactList* which is updated immediately following each periodic NTDS update.

B. COMMUNICATION WITH EXTERNAL SYSTEMS

We have identified three external agencies with which UNETGRAF must communicate--the user, the ATD/UNT Network Administrator and the host's NTDS system. The messages that flow across the interfaces between UNETGRAF and these external systems are shown in Figure 4.1. In order to more formally specify the exact nature of these messages, we have written them in a format based on Berzins' MSG84 specification language [BERZ85, BERZ86].

1. Messages from the User to UNETGRAF

MESSAGE ModifyNTDSDisplay(User:user_struct, Event:event_type)
TRANSITION by altering User to record Event

% The verb "TRANSITION" means that UNETGRAF undergoes a state
% transition in which some internal data structure is updated

MESSAGE ModifyNetworkOverlay(User:user_struct,
Event:event_type)
TRANSITION by altering User to record Event

MESSAGE ModifyConnectivityOverlay(User:user_struct,
Event:event_type)
TRANSITION by altering User to record Event

MESSAGE ModifyRoutingOverlay(User:user_struct,
Event:event_type)
TRANSITION by altering User to record Event

MESSAGE ModifyNodeDetailDisplay(User:user_struct,
Event:event_type)
TRANSITION by altering User to record Event

2. Messages between the Network Administrator and UNETGRAF

Network Administrator --> UNETGRAF

MESSAGE UpdateNodeList(NewNodeList)
TRANSITION by replacing the old NodeList with NewNodeList

MESSAGE UpdateConnectivityMatrix(NewConnectivityMatrix)
TRANSITION by replacing the old connectivity information with
NewConnectivityMatrix

MESSAGE UpdateRoutingTable(NewRoutingTable)
TRANSITION by replacing the old routing information with NewRoutingTable

UNETGRAF --> Network Administrator

MESSAGEGetStatistics(node_of_interest:node_report)
REPLY with node_of_interest:node_report
(statistics fields will have been updated)

3. Messages between the NTDS Translator and UNETGRAF

NTDS Translator --> UNETGRAF

MESSAGE UpdateContacts(NewContactList)
TRANSITION by replacing the old ContactList with NewContactList

MESSAGE UpdateOwnship(new_ownship_position)
TRANSITION by replacing old position with new_ownship_position

MESSAGE UpdateDLRP(new_DLRP)
TRANSITION by updating the DLRP (data link reference point)

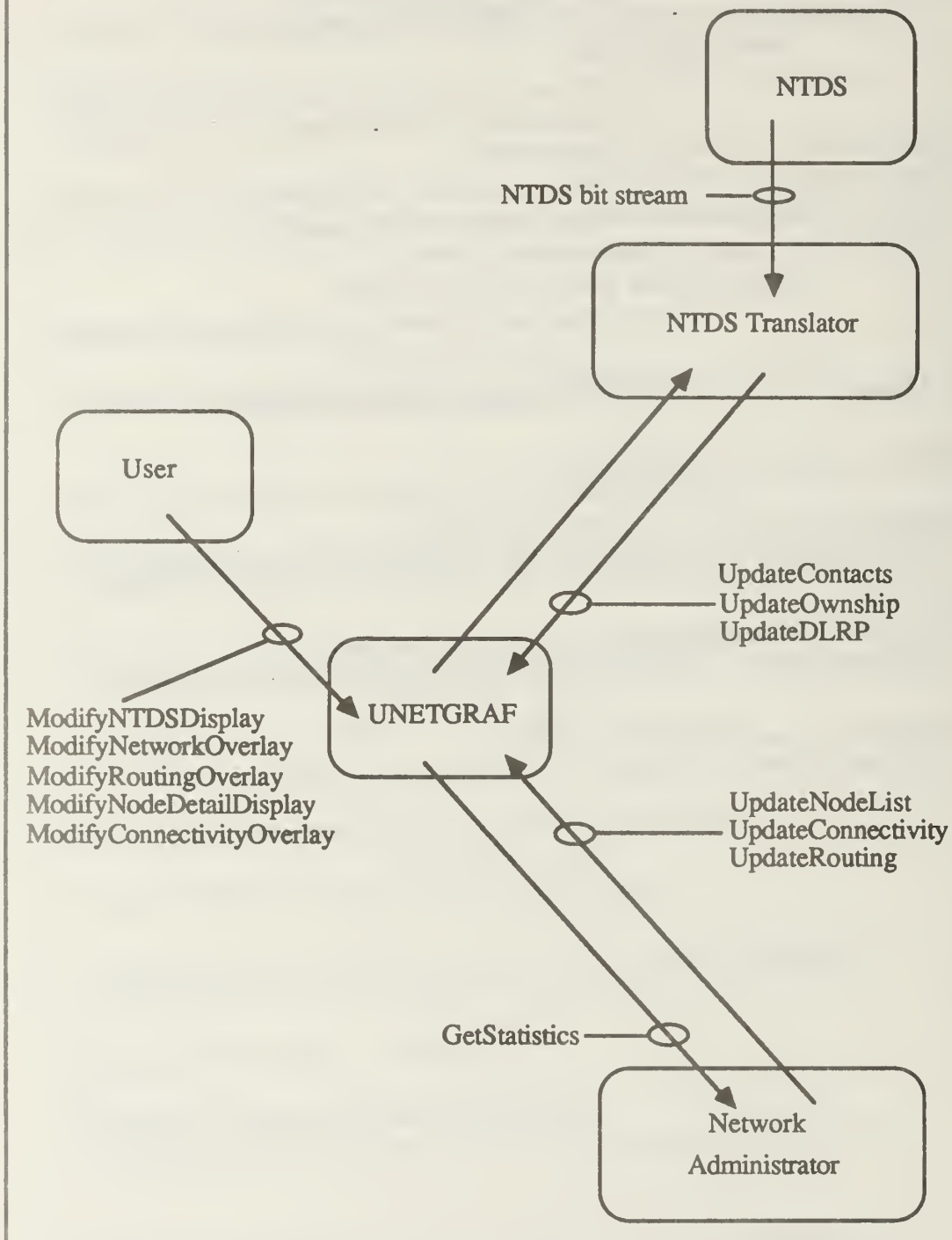

Figure 4.1 UNETGRAF Communication with External Systems

V. UNETGRAF ARCHITECTURAL DESIGN

In Chapter III we described the set of displays that we claim are appropriate for quickly conveying the state of the ATD/UNT network to UNETGRAF users. Chapter IV described UNETGRAF's interfaces with (1) the user, (2) the host's NTDS system and (3) the UNT Network Administrator. Our objective in this chapter is to describe the internal organization, or architectural design, used by UNETGRAF to convert the inputs described in Chapter IV into the output displays described in Chapter III. Figure 5.1 provides a high level view of the UNETGRAF architectural design. We proceed now to discuss each module/subsystem shown in this figure. The reader is urged to maintain a breadth-first orientation by frequent reference to Figure 5.1.

A. THE UNETGRAF.C MODULE

The *unetgraf.c* module drives the UNETGRAF program by performing simulated updates of the NTDS and network information and by dispatching user-generated events to the various event handlers. Figure 5.2 depicts the operation of this module as a sequence of calls to lower-level dependent modules.

B. THE USER INTERFACE SUBSYSTEM

The User Interface subsystem consists of three high-level event-handler modules, *menubutton.c*, *mousedown.c*, and *mouseup.c*, and dozens of lower level routines which use the IRIS pick mechanism, pop-up menu package, and *mex* window manager. As shown in Figure 5.2, all user-generated events are sent by *unetgraf.c* to the appropriate high-level module. There the *user_struct* record is updated to record the user interaction. As a result, the affected display is altered during subsequent drawing by the *draw_NTDS.c* and *draw_detail.c* modules. The IRIS pick mechanism and pop-up menu package provide the feedback facilities that enable these event handler modules to determine the appropriate response to mouse input. Since the *mex* window manager proved to have shortcomings for the UNETGRAF application, a modified window manager was designed and implemented. Its features are described in Appendix B (THE MODIFIED IRIS WINDOW MANAGER).

C. THE OBJECTS.C MODULE

All of UNETGRAF's drawing commands are called by the *objects.c* module. These commands are pre-compiled and stored in variables of type Object (one Object per window) for subsequent display in either the NTDS display window or a Node Detail window. Refer to Figure 5.3.

D. THE DRAW_NTDS.C MODULE

All drawing commands for the NTDS window emanate from this module. Included among these are the commands required to depict the *NetworkOverlay*, *ConnectivityOverlay*, and *RoutingOverlay* as well as those that draw the *TacticalDisplay*. Refer to Figure 5.4.

E. THE DRAW_DETAIL.C MODULE

All drawing commands for the set of currently open Node Detail windows originate in this module. Refer to Figure 5.4.

F. THE CORRELATE.C MODULE

The *correlate.c* module maintains a pair of cross-reference tables that permit an ATD/UNT node symbol to be matched (or "correlated") with its host's NTDS symbol and vice versa. Nodes whose hosts cannot be located in the *ContactList* are called "uncorrelated" nodes. The *correlate.c* module cooperates with the *contacts.c* module (described below) to store uncorrelated nodes as temporary elements of the *ContactList*. Refer to Figure 5.2.

G. THE TACTICAL DISPLAY STORAGE SUBSYSTEM

This subsystem consists of two modules. Refer to Figure 5.4.

The *contacts.c* module responds to messages from within UNETGRAF, providing (1) retrieval of *contact_reports* from the *ContactList*, (2) cooperation with the *correlate.c* module by storing uncorrelated nodes as temporary members

of the *ContactList*, and (3) animation of contact symbols by dead reckoning (DR) based on the contact's present course and speed.

The *NTDS_ifc.c* module communicates with the NTDS Translator to initialize and update the *ContactList*. In the current version, this module is a stub in which the *ContactList* is initialized from the text file "contact.record". NTDS updates consist of randomly derived course and/or speed changes for randomly selected *contact_records*.

H. THE NETWORK DISPLAY STORAGE SUBSYSTEM

This subsystem consists of four modules. Refer to Figure 5.5.

The *nodes.c* module provides (1) retrieval of *node_reports* from the *NodeList*, (2) retrieval of other network parameters; e.g., node quantity and (3) communication with the Network Administrator to obtain node performance statistics.

The *NA_private.c* module is a stub which initializes and updates the network. Initialization is begun by reading network parameters from the text file "network.record". The rest of the initialization process and the update process are identical and consist simply of updating network-wide adjacency matrices (one for each channel) with randomly derived channel values, and then calculating least-cost paths between all pairs of nodes in the network using a variation on Dijkstra's Shortest Path algorithm [HORO83].

The *NA_ifc.c* module communicates with the ATD/UNT Network

Administrator. In the present prototype version, it is a stub in which the initialization and periodic updates are accomplished by calling functions provided by the *NA_private.c* module.

The *NA_sim.c* module is a stub which provides randomly derived packet statistics in reply to the GetStatistics message. Refer to Figure 4.1.

I. THE UTILITIES SUBSYSTEM

This subsystem contains special-purpose graphics initialization functions and general purpose routines called from throughout the UNETGRAF program. Refer to Figure 5.6.

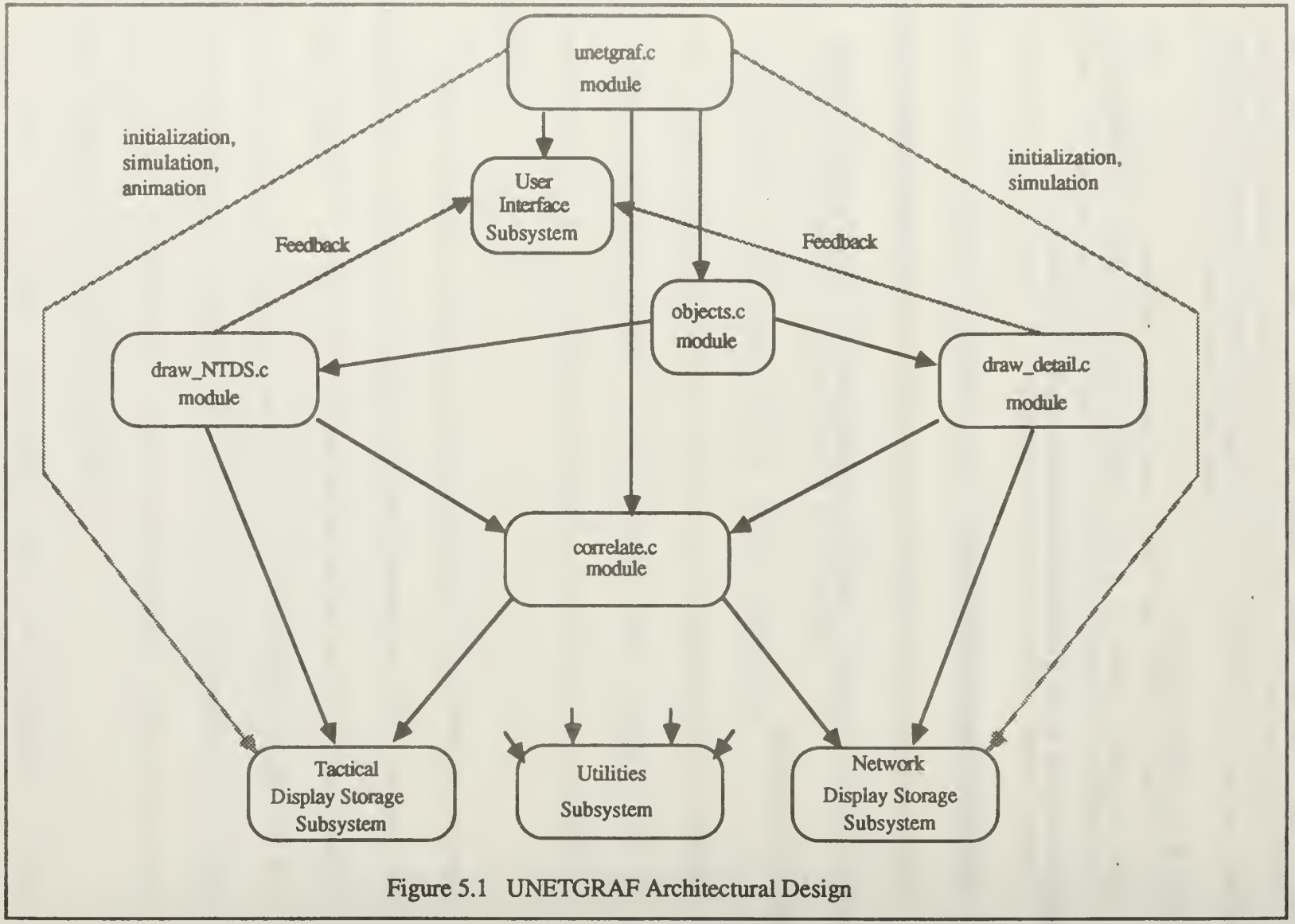

Figure 5.1 UNETGRAF Architectural Design

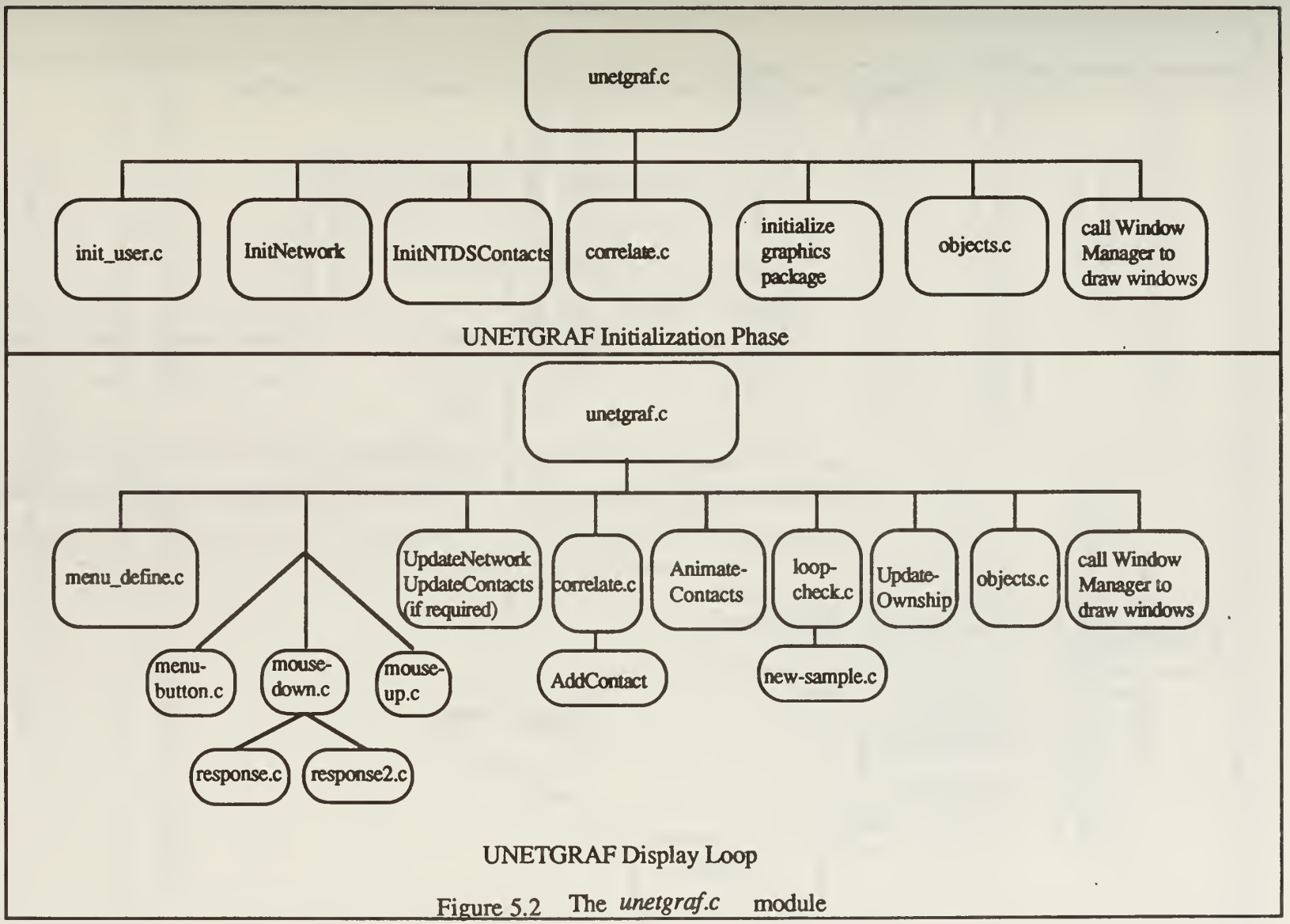

Figure 5.2 The *unetgraf.c* module

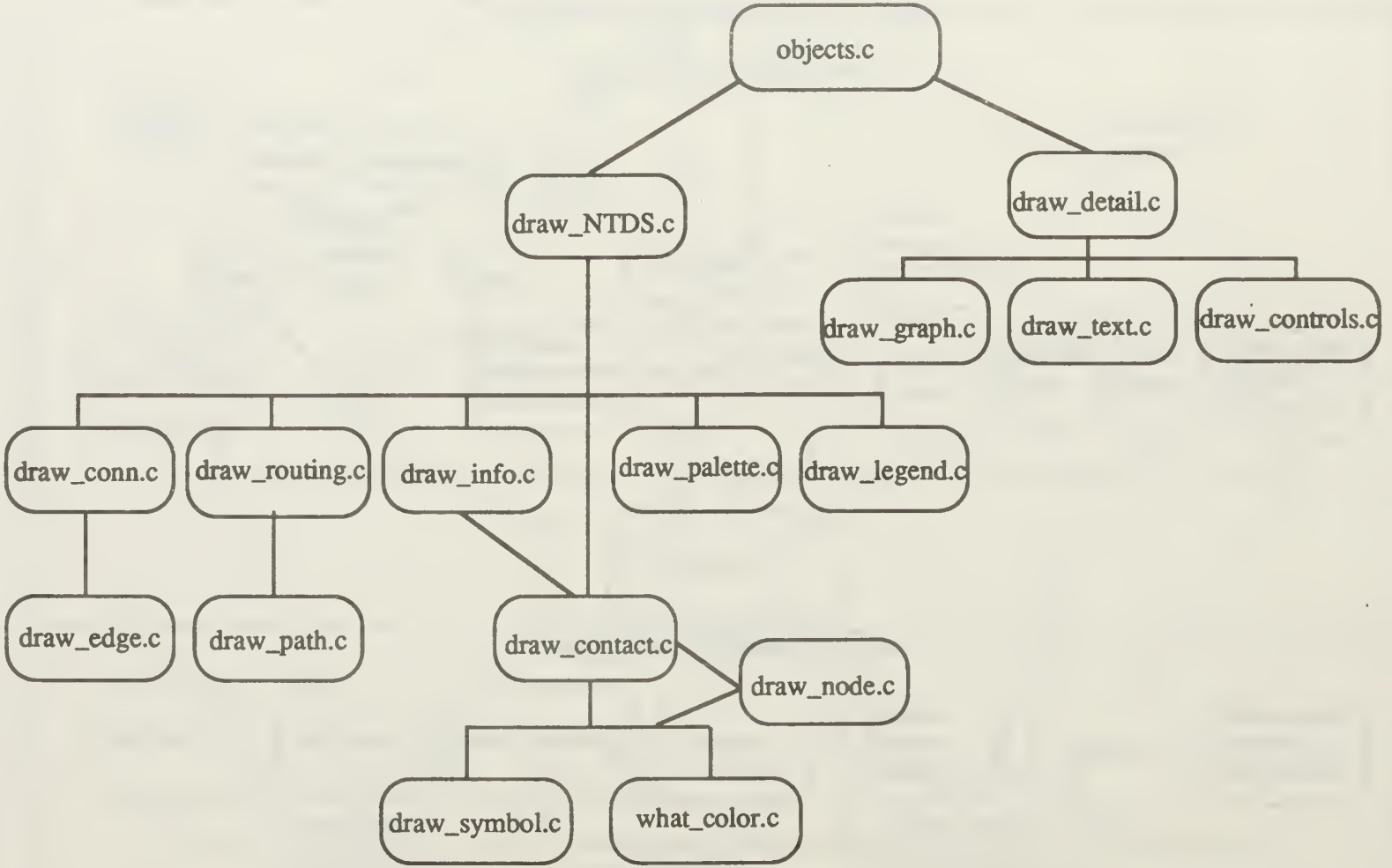

Figure 5.3 The objects.c, draw_NTDS.c, and draw_detail.c Modules

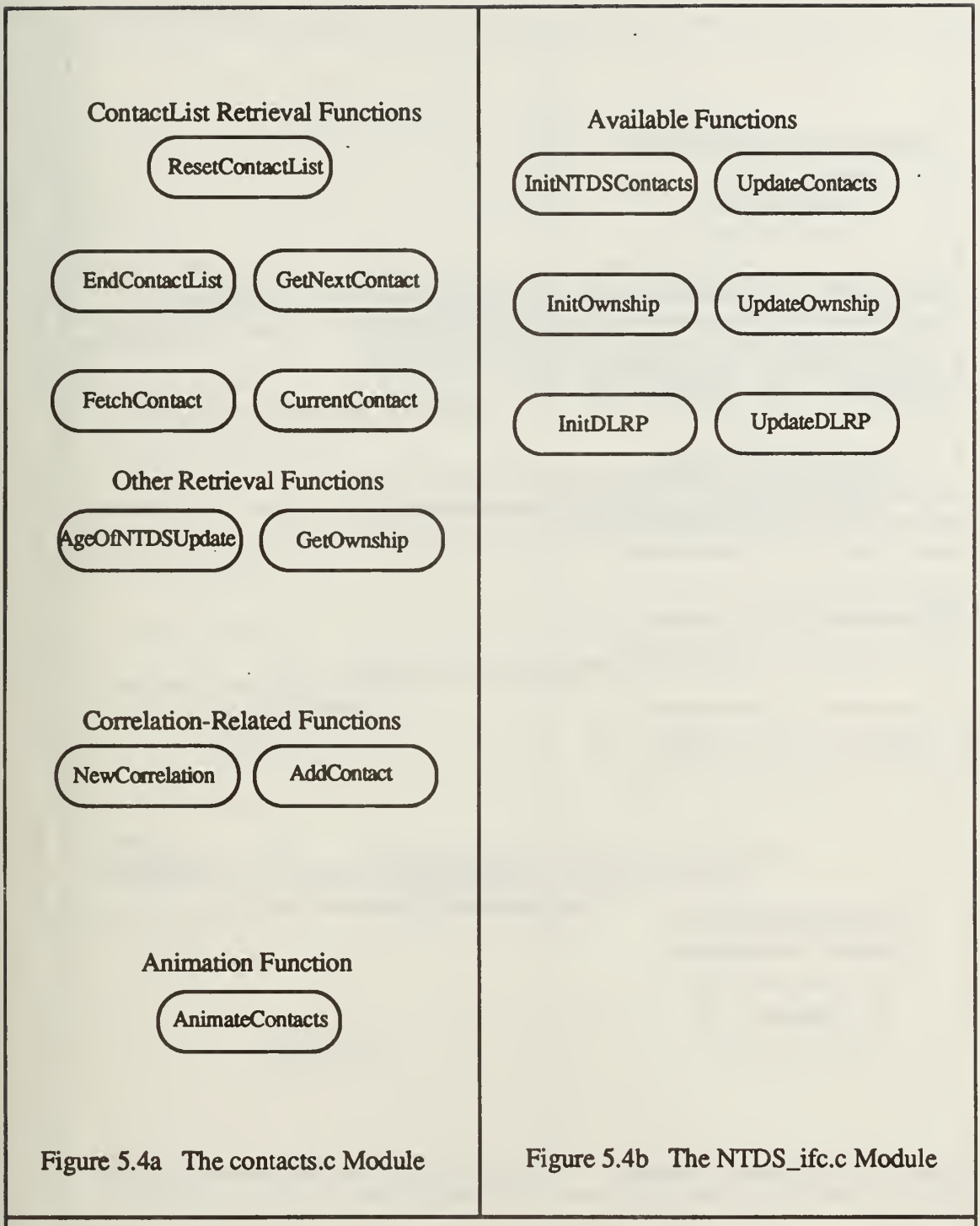

Figure 5.4a The contacts.c Module

Figure 5.4b The NTDS_ifc.c Module

Figure 5.4 The UNETGRAF Tactical Display Subsystem

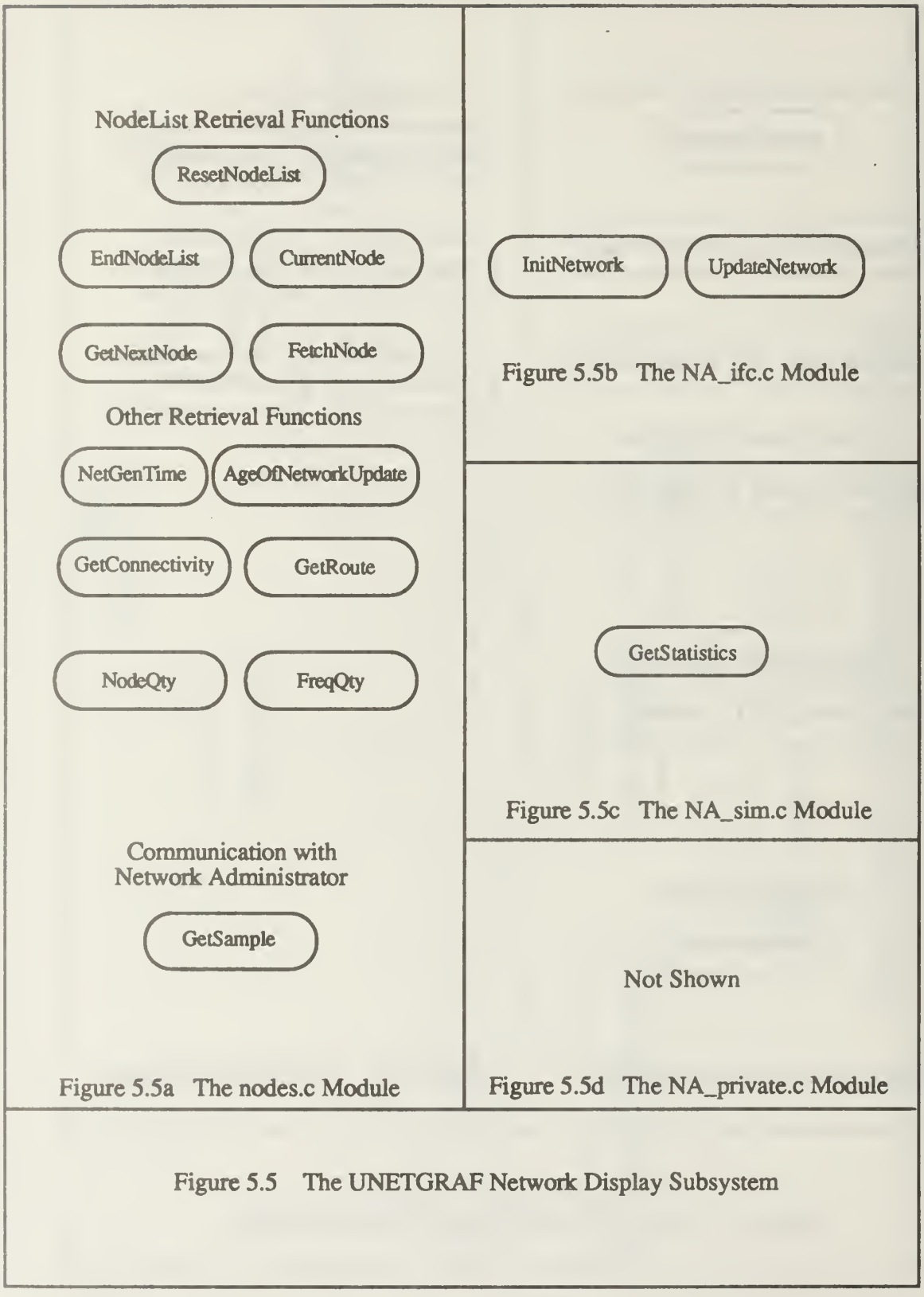

Figure 5.5a The nodes.c Module

Figure 5.5b The NA_ifc.c Module

Figure 5.5c The NA_sim.c Module

Figure 5.5d The NA_private.c Module

Figure 5.5 The UNETGRAF Network Display Subsystem

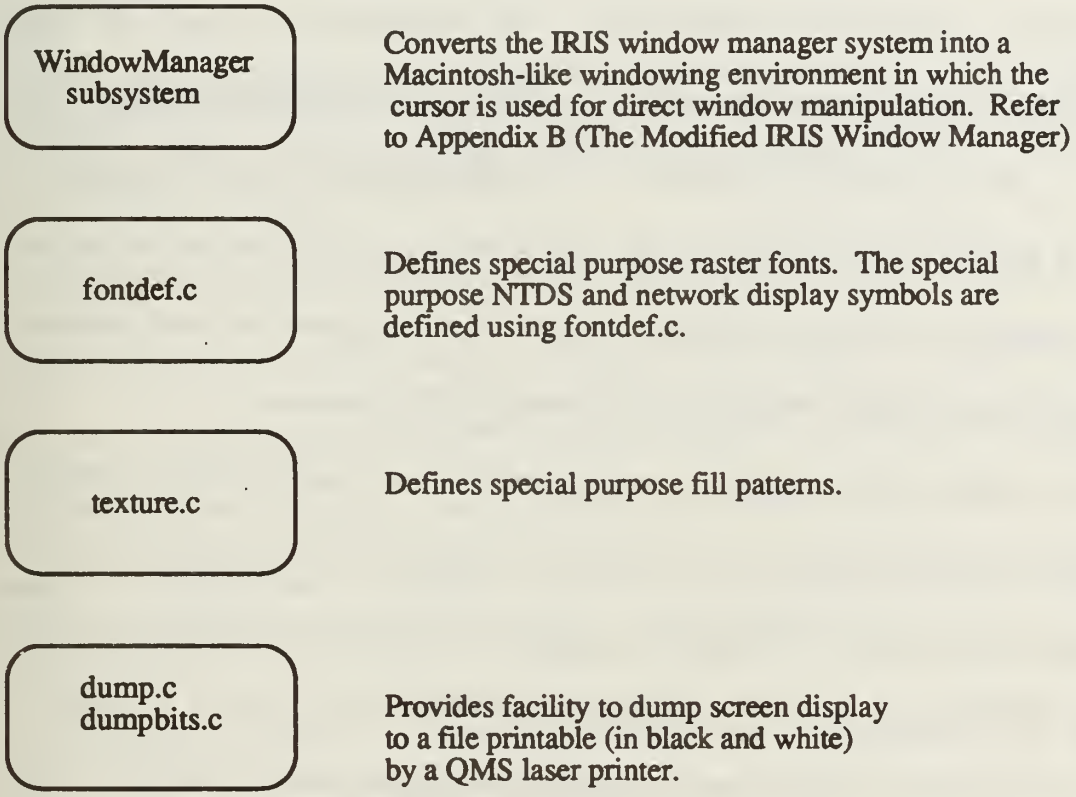

Figure 5.6 The UNETGRAF Utilities Subsystem

VI. SUMMARY AND CONCLUSIONS

We have used the software life cycle model as a framework for describing the requirements, functional specifications and architectural design of the UNETGRAF network monitor system. We believe the UNETGRAF design will prove to be exceptionally amenable to the evolution that will inevitably be required as the Advanced Technology Demonstration (ATD) project begins to more concisely describe and finally implement the various link and routing protocols for the Unified Networking Technology (UNT) network.

Our reading of the relatively sparse literature on graphics-based network monitors and on the formats for the interactive displays used in existing network monitors indicates that UNETGRAF is unique in its use of an established tactical display (NTDS) as a base for presenting information on the state of a network. As a class, network monitors tend to present information which is specific to that network for which they were designed. Apart from some high-level and decidedly abstract views of connectivity, the interactive displays that are provided by most network monitors are text-based and designed to be useful for trouble-shooters rather than users of the network [TERP82]. Since the UNT project has not proceeded far enough for specific trouble-shooting criteria to be defined, we have been forced to base our monitor on the network's higher level structures as represented by *ConnectivityOverlays* and *RoutingOverlays*. The use of the NTDS

tactical display as a base for these overlays has provided the UNETGRAF system with some immediately realizable benefits:

- The display format is familiar to those personnel that the UNT network is designed to serve, namely those officers and sailors within the naval battle group who are involved with command, control and communications functions.

- The effect of inter-node distance on network topology is, of course, immediately visible and, with additional input, the *TacticalDisplay* can also depict weather and land-forms, which have appreciable effects on HF and UHF radio propagation respectively.

- The possible sources of hostile jamming activity can be quickly inferred.

For the near term, UNT network designers are in largely the same role as early aircraft designers--the objective is simply to get the project off the ground and make it reliable enough to be accepted as *part* of fleet communications. We hope that UNETGRAF will play some role in the accomplishment of this objective; however, we do foresee the eventual resolution of all technical problems and the UNT network, or something similar to it, underlying *all* fleet communications at some point in the future.

Using this as a point of departure, we can speculate on the use of a UNETGRAF-like system as a network control device instead of a mere monitor. Replacing Node Detail windows would be Node *Control* windows. Therein would be interactive displays for adjusting transmitter power/receiver sensitivity, for enabling reserve channels, or possibly even for selecting a jamming or deception mode if the network has nodes aboard unmanned platforms. The close correlation

of the tactical situation and the communication situation that is part of UNETGRAF's displays will almost certainly characterize the battlefield of the future in which the electronic order of battle must be as well understood and as well-controlled as the air order of battle is today.

APPENDIX A – UNETGRAF USER'S GUIDE

A. ABOUT UNETGRAF

UNETGRAF is a computer graphics program that displays the status of a rapidly reconfigurable tactical communications network in real-time. The network involved is currently being developed by the Naval Ocean Systems Center (NOSC) under the Advanced Technology Demonstration (ATD) project of the Unified Networking Technology (UNT) program (short title: ATD/UNT). UNETGRAF was developed at the Naval Postgraduate School's Graphics and Video Laboratory and is currently implemented on the Laboratory's Silicon Graphics IRIS workstation.

B. ABOUT THE ATD/UNT NETWORK

UNETGRAF users are assumed to be familiar with the general aspects of computer and data communications and the goals of the NOSC UNT program; however, these will be briefly restated here.

Microcomputers are revolutionizing tactical communications networking by providing inexpensive storage and processing sufficiently powerful to implement store-and-forward capabilities at mobile communication nodes such as ships and aircraft. These capabilities permit the network to be aware of its own topology and automatically relay a messsage from any source node to any destination node.

The ATD/UNT project will demonstrate these capabilities by interposing a software layer called a Multinetwork Controller (MC) between the battle group's communications clients--those persons and systems using radio communications--and the battle group's communications servers--the radio equipment that performs the actual transmission and reception. Figure A.1 shows the configuration of a communications node under the ATD/UNT concept.

Ideally, the Multinetwork Controller would be able to call on its database (known as the Network Administrator (NA) in the ATD/UNT network) for information about the entire network's topology. Such information would enable the MC to make near-perfect decisions about routing. However, the exigencies of naval combat require that each node be able to make routing decisions independently. Thus, unless a large part of the ATD/UNT network's available bandwidth is given over to sharing network management information among nodes, the MC at a given node will know only about its nearest neighbors. Additionally, since all network nodes are hosted by ships or aircraft which are continually moving, even this information is perishable. The MC must, therefore, periodically update adjacency information and, while doing so, share routing information with its neighbors. In networking parlance, such a network is said to possess a distributed, adaptive routing protocol.

C. UNETGRAF DISPLAYS

Though a network such as ATD/UNT will necessarily have a good deal of *internal* complexity, its external behavior can still be modeled as a set of adjacencies for each node and a set of (possible empty) paths between all pairs of nodes. UNETGRAF's displays depict these external characteristics of the ATD/UNT network with displays that will be referred to as the *NetworkOverlay, ConnectivityOverlay* and *RoutingOverlay*. Additionally, a *NodeDetailDisplay* is provided to depict such items as packet counts, error statistics and idle capacity. In order to orient the user and depict the network in a proper geographical context, UNETGRAF makes use of the node's host's Naval Tactical Data System (NTDS) database to construct an underlying *TacticalDisplay*. If the host's NTDS system is not available, UNETGRAF defaults to a network-only mode in which the *NetworkOverlay* is drawn over an empty *TacticalDisplay*.

D. UNETGRAF CONTROLS

UNETGRAF accepts input from pop-up menus (activated by the right mouse button) and selection of on-screen controls by the left mouse button. Menu configurations are shown in Figure A.2. Menu options are selected by releasing the right mouse button with the desired option highlighted. Menu input affects only the *TacticalDisplay*. Some additional *TacticalDisplay* controls are provided by the Cursor Palette (Figure A.3) but this control device exists primarily to manipulate the network-related overlays. A new cursor is selected from the Cursor

Palette by placing the existing cursor over the desired selection and clicking the left mouse button.

E. INITIALIZING UNETGRAF

At this writing, UNETGRAF is an isolated system. The Network Administrator is not present to provide information on the state of the network nor is a host ship or aircraft available to provide NTDS information. UNETGRAF must therefore obtain initial data from two text files as described below.

The "contact.record" file contains a list of *contact_reports* which defines the set of NTDS contacts to be displayed. If this file is not present, UNETGRAF defaults to a network-only mode.

The "network.record" file contains the declarations of (1) total nodes in the network (including leaf nodes), (2) number of branching nodes (these provide UNT's store-and-forward capabilities), (3) number of communication channels available to the branching nodes, and (4) a "connectivity adjustment factor" (1.0 = a richly connected network; 0.0 = a network with no connections). If this file is not present, UNETGRAF uses default values for each of these declarations.

Samples of the "contact.record" and "network.record" files are contained on npscs-unix1:/work2/griggs/unetgraf.

F. STARTING UNETGRAF

At the system prompt, type

mex <CR>

This invokes the IRIS *mex* window manager. When the system prompt is again visible, type

unetgraf <CR>

After a few seconds, the initial display, entitled "UNETGRAF NTDS DISPLAY", will be visible.

G. THE UNETGRAF NTDS DISPLAY

1. General

UNETGRAF starts up in a single fixed window entitled "UNETGRAF NTDS DISPLAY" with only the *TacticalDisplay, NetworkOverlay* and Cursor Palette visible. The *TacticalDisplay* (Figure A.4) is a Naval Tactical Data System (NTDS) display with Advanced Combat Direction System (ACDS) symbols [NAVS83] that uses color and symbol shape to redundantly denote the allegiance of a contact. An "information box" (containing contact course, speed, etc) can be obtained by positioning the "Arrow" cursor over the desired contact and clicking the left mouse button. The information box can be closed by clicking on it with any cursor from the Cursor Palette. Unlike a standard NTDS display, the contacts depicted here move *between* updates, based on last course and speed.

This feature is called "DR" (for dead reckoning). To disable the "DR" feature, select "Disable DR" from the NTDS Display menu.

The *NetworkOverlay* (Figure A.5) consists of node symbols (triangles) that overlie those contacts that have been identified as hosts for ATD/UNT nodes. If a node's host cannot be identified, the node symbol remains visible, but does not overlie an NTDS symbol. Such a node is said to be "uncorrelated". If no NTDS positioning data is available, the *NetworkOverlay* defaults to the "network-only mode". Figure A.6 shows this mode. Occasionally, a node symbol may begin to flash at a one Hz rate. This action, known as a "high use alarm", is an indication that the node's idle capacity is less than five percent of its total capacity. This high use alarm can be disabled by clicking on the flashing node with the "Eraser" cursor.

2. Adding Overlays to the Initial Display

The *ConnectivityOverlay* (Figure A.7) is a directed graph with vertices at those node symbols designated by the "Connectivity" cursor. The edges of this graph (represented by solid white lines with arrowheads) depict the adjacencies (i.e., immediate neighbors) of the designated node. A node symbol can be removed from the *ConnectivityOverlay* by a left mouse click with the "Eraser" cursor.

The *RoutingOverlay* (Figure A.8) consists of a single designated source node and any number of designated destination nodes. The path from the source node to each destination node is shown as a dashed line. The source node is

designated by the "Source" cursor and identified by a surrounding square. Destination nodes are designated by the "Destination" cursor and identified by reverse video within a surrounding square. Both designations can be removed by a left mouse click with the "Eraser" cursor.

H. UNETGRAF WINDOWS

The *TacticalDisplay*, *NetworkOverlay*, *ConnectivityOverlay*, and *RoutingOverlay* are all displayed in the fixed opening window. This window cannot be moved, reshaped or closed; however, additional windows containing *NodeDetailDisplays* can be manipulated by the user once they are opened. The control regions of these windows are shown in Figure A.9. These control regions are used as follows:

To close a window:

Position the cursor in the go-away box and click the left mouse button. The window will disappear.

To move a window:

Position the cursor in the drag region. Hold down the left mouse button and drag the cursor (which now is a "move" icon) to the desired location. Release the left mouse button and the window will be redrawn in the new location.

To resize a window:

Position the cursor in the reshape region. Hold down the left mouse

button and drag the cursor (which now is a "reshape" icon) to the desired location. Release the left mouse button and the window will be redrawn with the new dimensions.

To pop a window:

Position the cursor anywhere in the content region. Click the left mouse button and the window will be redrawn at the top of the stack.

I. THE NODE DETAIL DISPLAY

1. General

The *NodeDetailDisplay* (Figure A.10) depicts activity at a node over a user-specified time interval. Each channel's packet counts during the interval are categorized as Voice, NTDS, TTY, Mgmt (network management messages), Bad (requiring NAK/retransmission) or Aged (exceeded hop count). Total counts in each category are provided for both the "transmit" and "receive" sides of each frequency. Idle capacity is also calculated and displayed for each channel and for the node as a whole.

2. NodeDetailDisplay Controls

a. Opening a New NodeDetailDisplay

Each *NodeDetailDisplay* is contained in a separate window. The window is opened by first clicking on the desired node with the "Arrow" cursor to obtain an "information box", then clicking on the dogear in the lower right corner of the information box with any cursor. The *NodeDetailDisplay* window will

either open immediately or a beep will sound, indicating that no more windows are available from the IRIS window manager.

b. Manipulating the NodeDetail Window

See the above section on UNETGRAF windows.

c. Altering the NodeDetailDisplay

The information in the *NodeDetailDisplay* window can be displayed either graphically (as a histogram containing percentages) or textually (as a table of packet counts) by clicking on either "Graph" or "Text" for each frequency. The three sampling options available are "Last sample taken", "Cumulative stats" and "New stats every *nn* seconds", where *nn* is the present sample interval. The sample interval, set by default to 60 seconds, can be modified by placing any cursor on the desired arrow (up or down) and pressing the left mouse button. Sampling options are selected by clicking on the desired text string with the left mouse button.

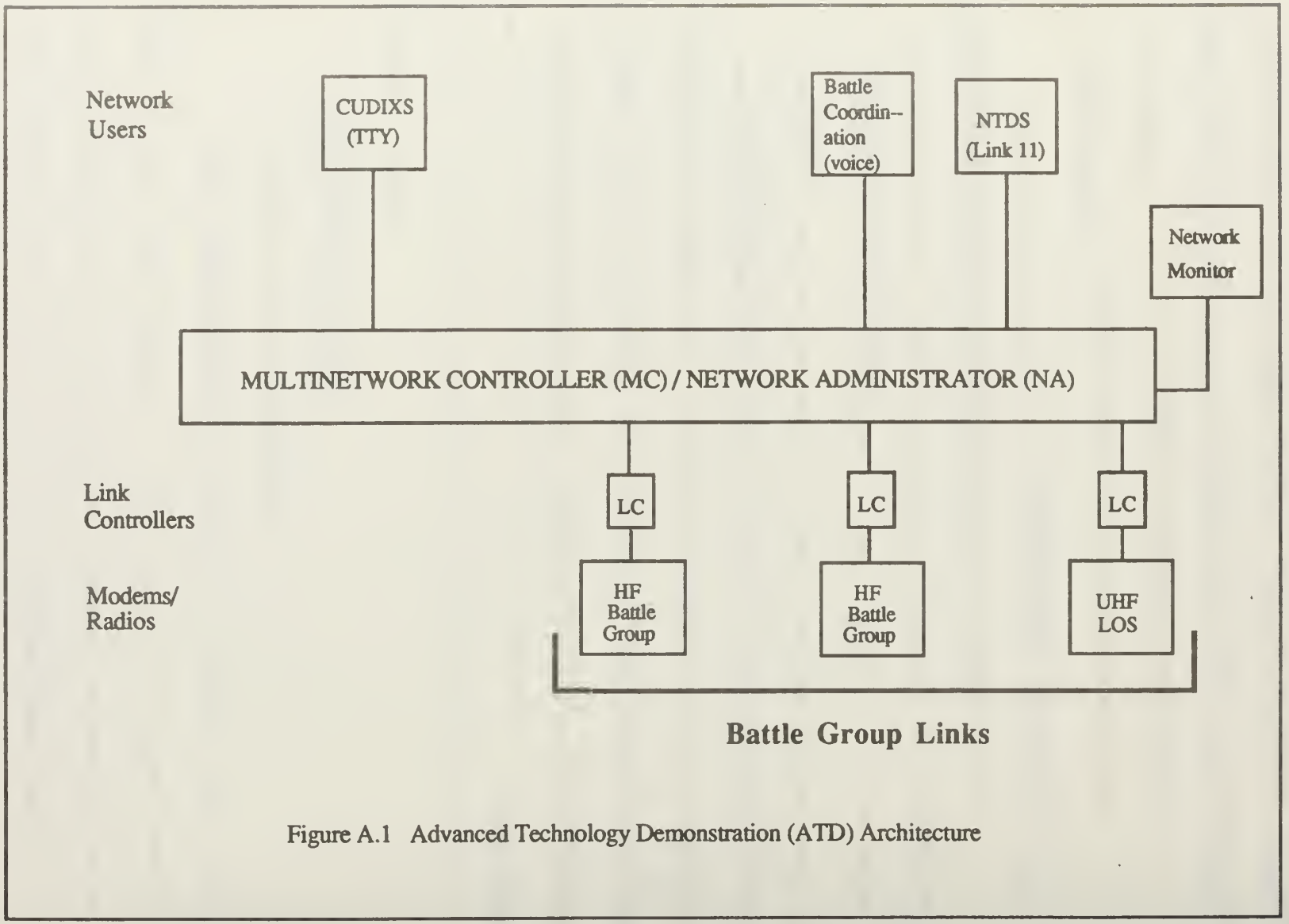

Figure A.1 Advanced Technology Demonstration (ATD) Architecture

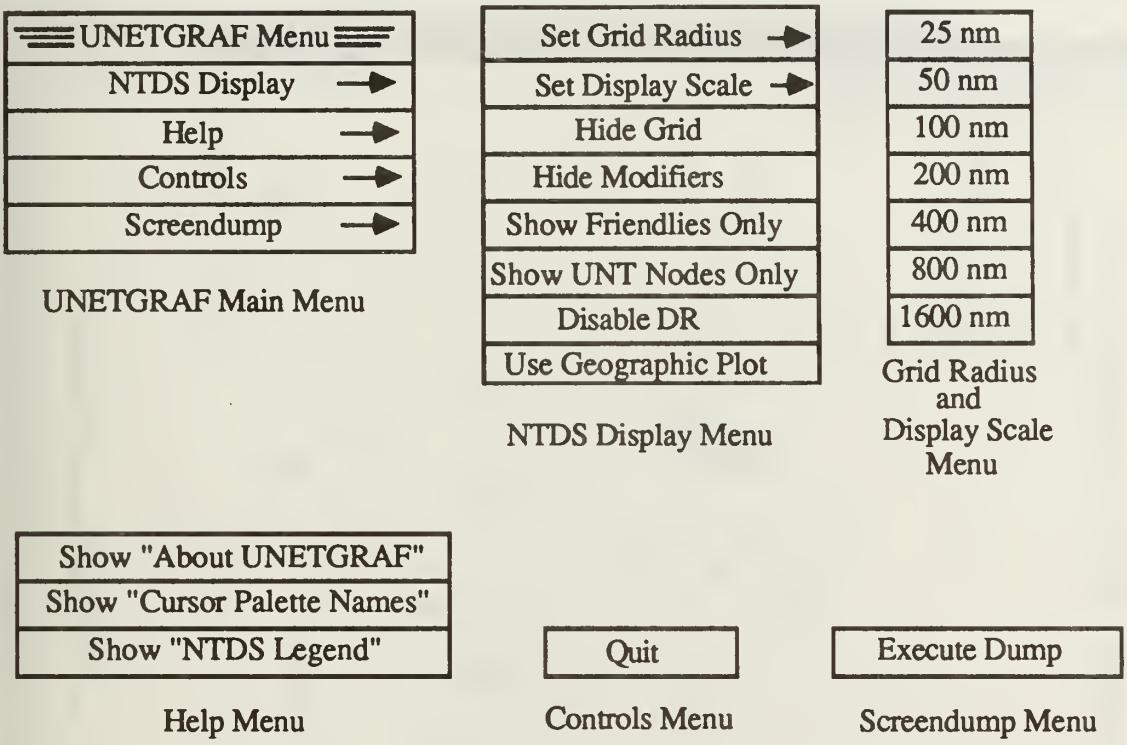

Figure A.2 UNETGRAF Menu Options

Figure A.3 The UNETGRAF Cursor Palette

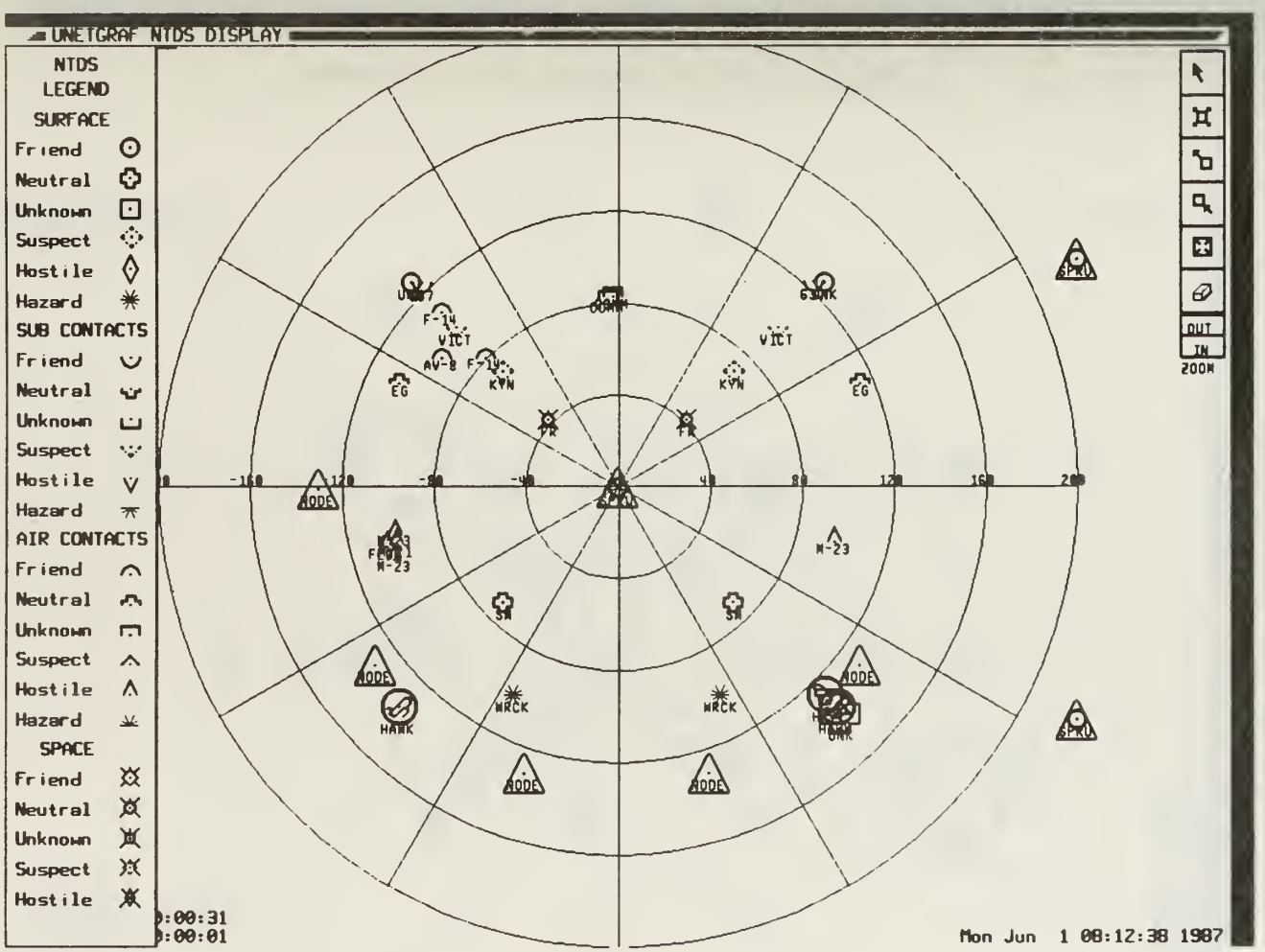

Figure A.4 The UNETGRAF Tactical Display

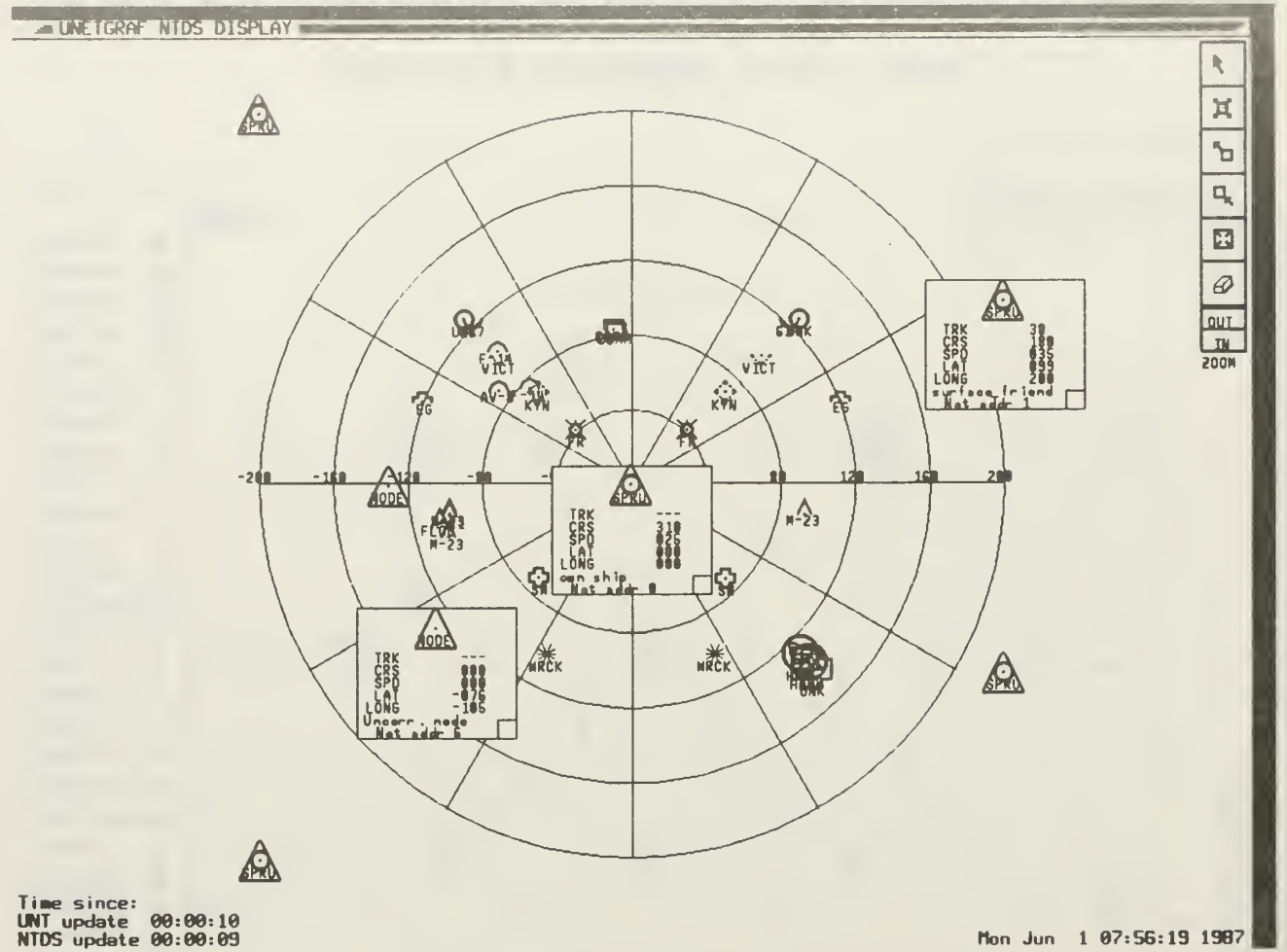

Figure A.5 The UNETGRAF Network Overlay

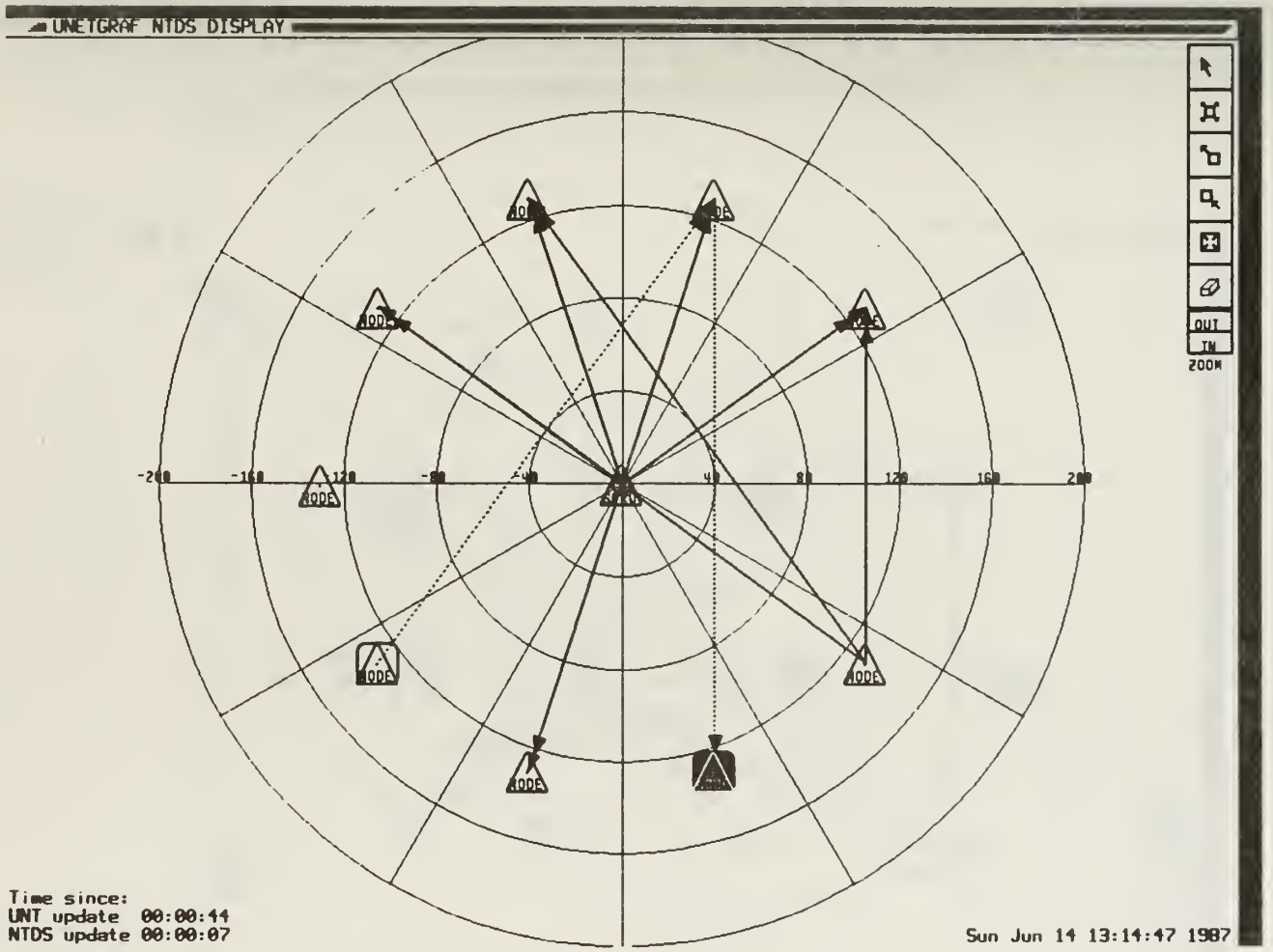

Figure A.6 A Network-Only Display

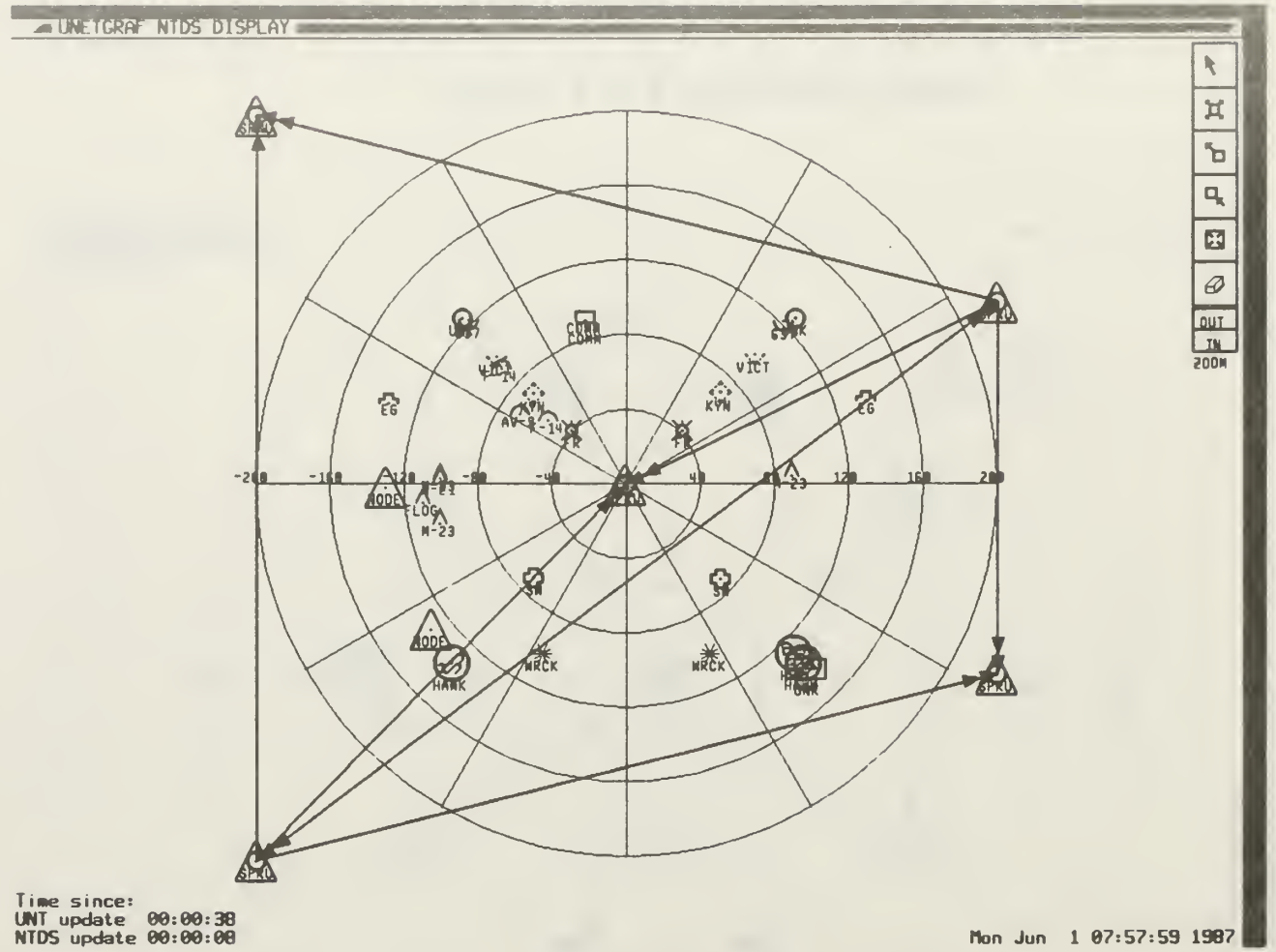

Figure A.7 The UNETGRAF Connectivity Overlay

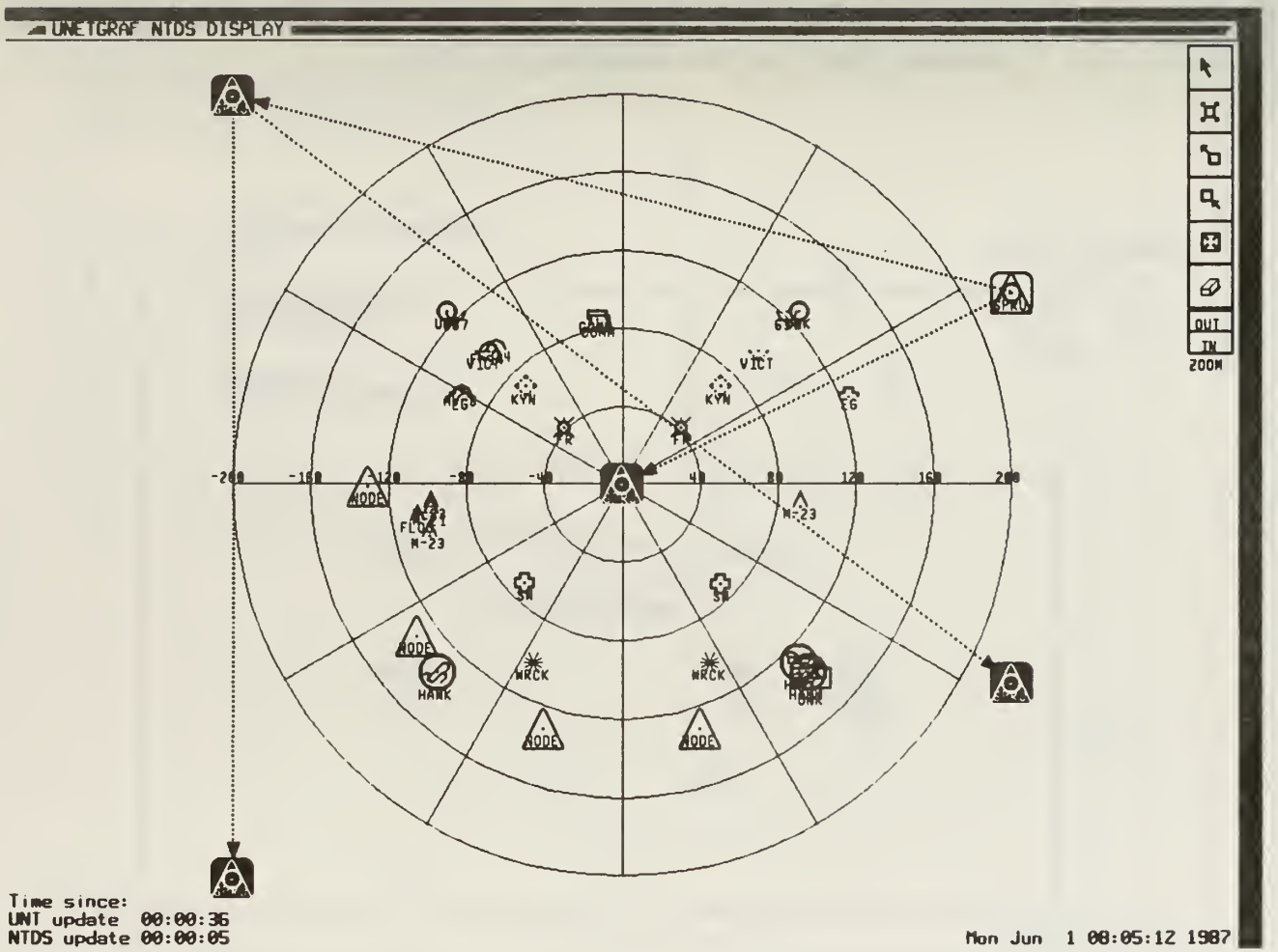

Figure A.8 The UNETGRAF Routing Overlay

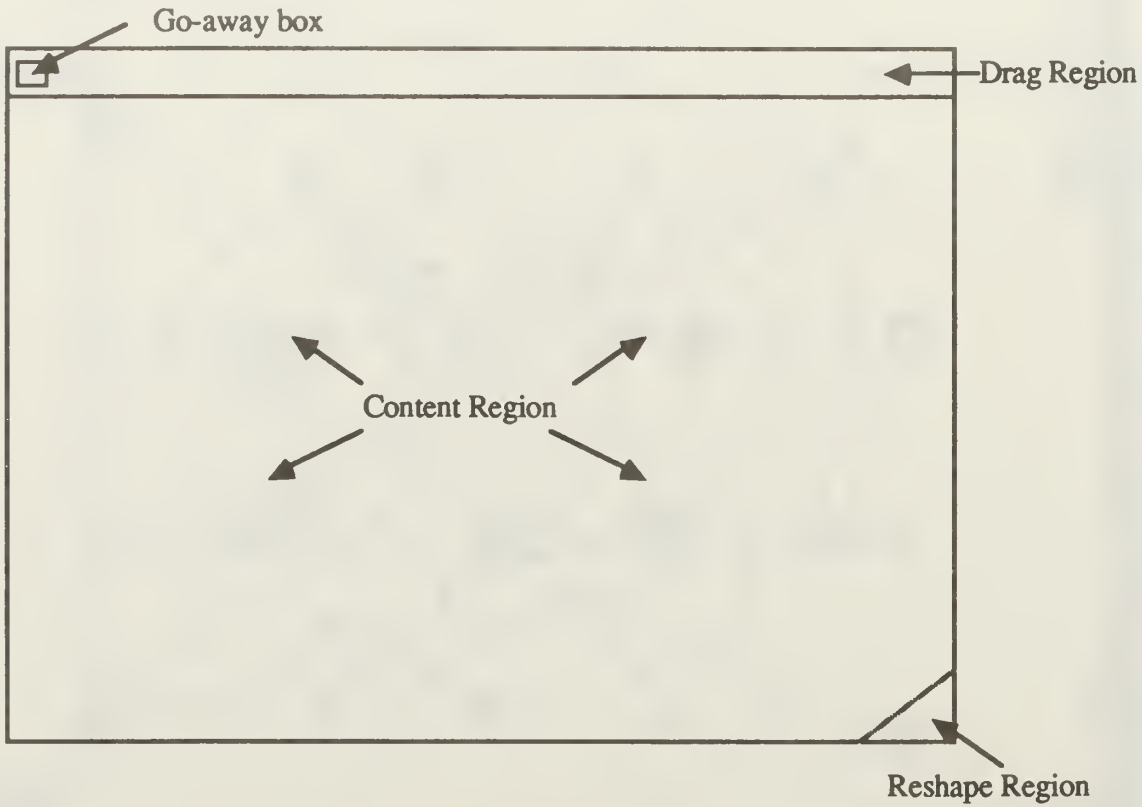

Figure A.9 Regions of the UNETGRAF Window

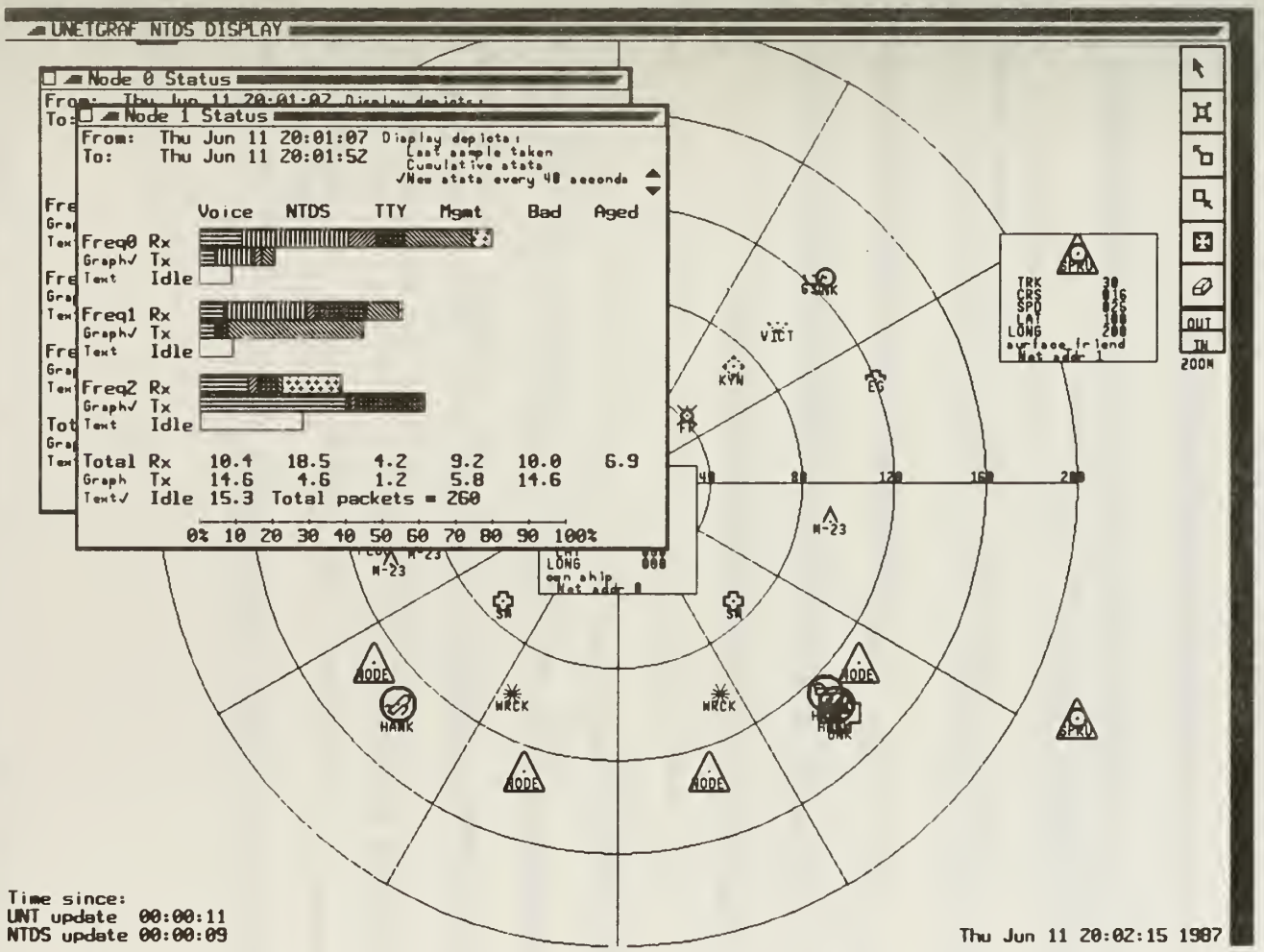

Figure A.10 The UNETGRAF Node Detail Display

APPENDIX B – THE MODIFIED IRIS WINDOW MANAGER

A. PURPOSE

The native IRIS window manager, *mex,* presents two drawbacks for programmers working on multi-window interactive graphics applications:

- It does not respond directly to mouse input like the familiar Macintosh or GEM window managers* ; instead, it is menu-driven.

- It provides no function to return the window in which a mouse event occurred.

The Modified IRIS Window Manager (MIWM) corrects these shortcomings with the goal of making the tasks of using and programming multiple window applications on the IRIS as much like the Macintosh as possible.

B. HOW MIWM WORKS

For the most part, MIWM uses the functions explained in the IRIS User's Guide Window Manager Appendix Section 2.3 (Changing Windows Non-interactively) and the user's mouse input to open, drag, pop and close windows. (Pushing windows (to the bottom of the window stack) is not supported by MIWM). MIWM uses a "private" data structure to keep track of the window stack.

* Macintosh is a trademark of the McIntosh Laboratory, Inc., licensed to Apple Computer, Inc. GEM is a trademark of Digital Research, Inc.

C. SOME CAVEATS

Because MIWM redraws entire windows on many occasions, programmers are required to treat the entire picture displayed in a window as a single graphical Object whose address must be provided as an argument when calling **OpenWindow**. (See the IRIS User Guide for additional information on Objects.)

In general, any application operating under MIWM (or *mex)* should reserve the right mouse button for *mex* operations. The MIWM designer's recommendation is that the left mouse button be used for picking and the right mouse button be reserved for *mex* pop-up menu interaction.

mex has a bug that MIWM has inherited. The IRIS User Guide Window Manager Appendix states that a single-process multiple-window application can have up to ten (10) windows open simultaneously; however, when more than five (5) are open simultaneously *mex* will sometimes (without any discernible pattern) not re-grant a graphics port after a window has been closed. The result is that the application program soon runs out of graphics ports. This bug has not occurred when the number of simultaneously open windows is limited to five (5) by the application programmer.

MIWM uses the top 16 rows of pixels in each window for control regions and (optional) title.

Programmers using MIWM should **not** use any native *mex* functions except those contained in Section 2.2 (Setting Constraints on Windows) of the IRIS User Guide Window Manager Appendix.

Programmers who find these constraints unacceptable will have to forego use of MIWM and use the native IRIS window manager routines instead.

D. THE MIWM WINDOW

The four regions of a MIWM window are shown in Figure B.1. As on the Macintosh, the go-away box is used to close the window, the drag region is used to move the window around the screen, the reshape region is used to resize the window and the content region is used as the programmer desires.

MIWM also provides a "background" window for those applications that require a window that the user cannot close, pop, move or resize. This "background" window does not have a go-away box, drag region or reshape region. Refer to the MIWM routine **OpenWindow** for more information.

E. USING MIWM (THE USER'S PERSPECTIVE)

1. Startup.

Programs using MIWM are, in fact, operating under *mex*. Therefore, before starting up the application program, type

mex <return>

at the system prompt. If you forget, MIWM will terminate the application program with a reminder.

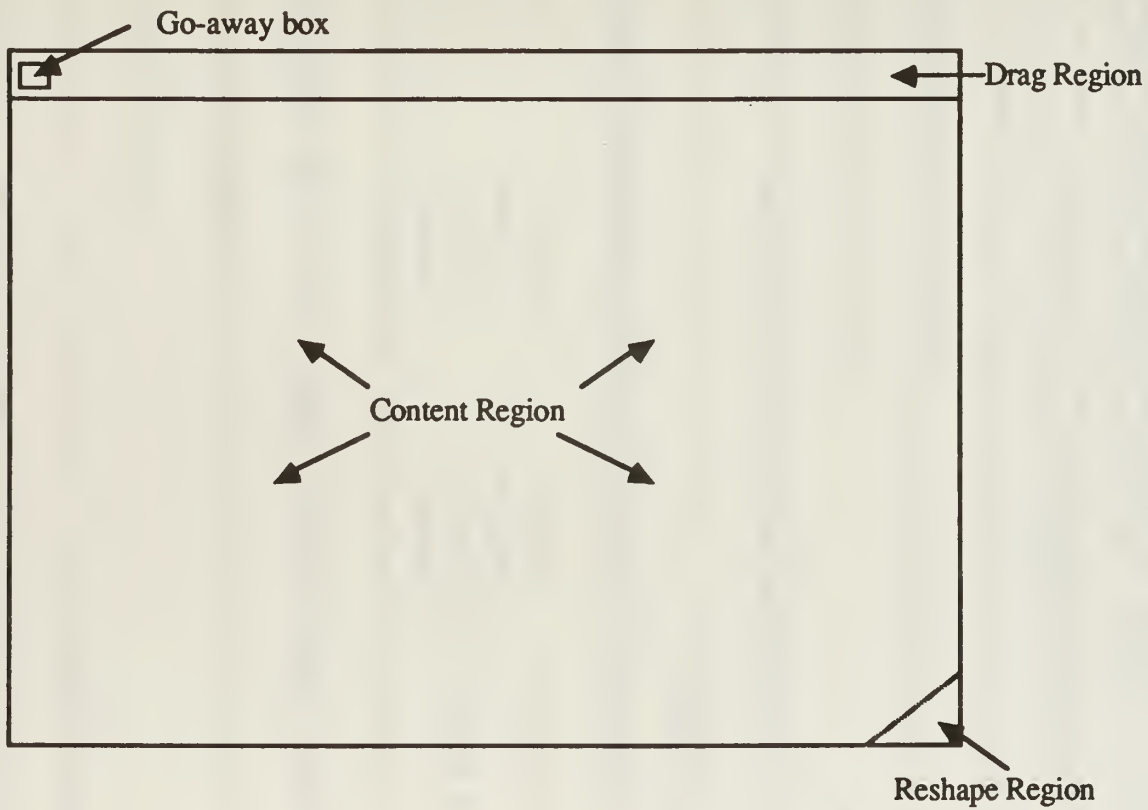

Figure B.1 Regions of the UNETGRAF Window

2. Opening a Window

Unless the programmer has fully specified both window location and size beforehand, you must do so.

You will be prompted by the cursor, which will have changed to an inverted "L"--an icon representing a corner of the new window.

Position the cursor at the desired location then hold down any mouse button and drag the cursor diagonally across the screen until the window outline is the desired size.

Release the mouse button and the window will appear.

3. Closing a Window.

Position the cursor in the go-away box and click the left mouse button. The window will disappear.

4. Moving a Window.

Position the cursor in the drag region. Hold down the left mouse button and drag the cursor (which now is a "move" icon) to the desired location. Release the left mouse button and the window will be redrawn in the new location.

5. Reshaping a Window.

Position the cursor in the reshape region. Hold down the left mouse button and drag the cursor (which now is a "reshape" icon) to the desired location. Release the left mouse button and the window will be redrawn with the new dimensions.

6. Popping a Window.

Position the cursor anywhere in the content region. Click the left mouse button and the window will be redrawn at the top of the stack.

NOTE TO PROGRAMMERS: Except for opening a window, MIWM routines must be explicitly called to accomplish all the above. Refer to the sample program *wintest.c* and to the next section for more information. Standardization of the user interface as described above is strongly encouraged.

F. PROGRAMMING WITH MIWM

1. Compile and Link Requirements

Programmers must #include the header file *windows.h* in any file that calls a MIWM function or contains a variable of type Window. Programmers must also link with the following fourteen (14) object files that comprise MIWM.

winclose1.o	winclose2.o	windrag.o
winfind.o	winlist.o	winopen.o
winpick.o	winpop.o	winregions.o
winsearch.o	winset.o	winshape.o
wintake.o	wintitle.o	

Source code files (for all the above object files) and *windows.h* are available on npscs-unix1:/work2/griggs/winmgr.

2. Program Structure

Programs using MIWM should follow the usual conventions of event-driven programming. A sample "shell" program, *wintest.c,* illustrates the use of all the MIWM functions and includes detailed in-line comments. Programmers are strongly encouraged to study the general program structure, in-line comments,

and actual operation of this program before attempting to write their own applications.

3. The MIWM WindowList

MIWM maintains a list of the programmer's currently open windows as a "private" data structure. The programmer can add windows to this list using **OpenWindow**; he can delete windows using **CloseWindow**; and he can get all the Window records in the list sequentially by using **ResetWindowList**, **EndWindowList**, **CurrentWindow** and **GetNextWindow** as shown in *wintest.c* and described below.

4. The MIWM Window Record.

The MIWM WindowList contains records of type Window which is declared as:

```
/* structure used to record information on each window */
typedef struct
{
   /* programmer provides these fields when he opens a window */
   char        title[50];/* title to be displayed in title bar */
   Object      *obj;     /* ptr to the object to be drawn in window */
   short       BackgroundFlag;
                /* boolean: does window always stay in
                   background?
                   If so, it can't be moved, popped
                   or resized */

   /* programmer uses these for any purpose he likes */
   long        refCon1,refCon2;

   /* MIWM keeps these fields current */
   int         wid;      /* IRIS-mex-provided unique window id (aka gid) */
```

```
    long        wx,wy;    /* x,y dimensions of the window */
    long        orgx,orgy;/* screen coordinates of lower left corner of window*/
} Window;
```

---

5. MIWM Routines

Task	Routine
"Attach" input focus to your program	TakeInputFocus
Open a new window	OpenWindow
Close an existing window	CloseWindow
Did user release the mouse button while in a go-away box	TrackGoAway
Pop a window to the top of the on-screen stack	PopWindow
Move a window under user control	DragWindow
Resize a window under user control	Reshape Window
Designate a particular window as the target of drawing command	SetWindow
Find out in which window and in what region an event occurred	FindWindow
Determine which window has a particular graphics identifier (gid)	WhichWindow
Write the window title in the window's title bar	DrawTitleBar
Get the window records of all the currently open windows	ResetWindowList, EndWindowList, CurrentWindow, GetNextWindow

G. FUNCTION SPECIFICATIONS

TakeInputFocus()

TakeInputFocus designates your program as the one to receive all input from mouse, keyboard and buttonbox.

```
Window     *OpenWindow(title,obj,BackgroundFlag)
char       title[];
Object     *obj;
short      BackgroundFlag;
```

OpenWindow adds another window to the MIWM WindowList, then calls the native IRIS routines that permit the user to set window size and location (unless already completely specified by the programmer). Since some of these native routines do the graphics initialization, the programmer can make **no** calls on any IRIS graphics library routines until after he calls **OpenWindow** for the first time. Refer to the sample program *wintest.c* for additional information.

Arguments to **OpenWindow** are the title string, address of the object containing all the drawing commands to be executed in the window, and a boolean indicating whether the window is a "background" window. Prior to calling **OpenWindow**, the programmer should call one of the following *mex*- provided routines to set initial window constraints:

- when opening a "background" window:

  Call **prefposition** with the desired window position.

- when opening a normal window:

  If you want the window to appear immediately, call **prefposition** with the

desired initial window position. The user will still be able to move and resize the window.

If you want the user to interactively specify the window's size and position, call **minsize(50,50)**. This will allow the user to provide initial size and position by means of the cursor. This size (50X50 pixels) will ensure that the window is large enough that even a novice user can keep track of it.

OpenWindow returns a pointer to the window record for the newly opened window. This gives the programmer the opportunity to update the two refCon fields within the window record if he has elected to use these fields. If no graphics ports are available, **OpenWindow** sounds the terminal bell and sends an exception string to stdout.

```
CloseWindow(theWindow)
Window    theWindow;
```

CloseWindow deletes a window record from the MIWM WindowList, then calls the native IRIS routines that do the required redrawing. Prior to calling **CloseWindow**, programmers should call the MIWM routine **TrackGoAway** to ensure that closing the window is what the user really had in mind. If an application program has only one window open, **CloseWindow** will consider any attempt to close this window as an exception, sounding the terminal bell and

sending an exception string to stdout. An unrecoverable system error would otherwise result.

```
int     TrackGoAway(goAwayWindow)
Window  goAwayWindow;
```

TrackGoAway returns TRUE if the user released the mouse button while the cursor was inside the go-away box; otherwise it returns FALSE.

```
PopWindow(theWindow)
Window    theWindow;
```

PopWindow modifies the MIWM WindowList to reflect the change in the on-screen stack of windows, then calls the native IRIS routines to do the actual popping and redrawing. The standard MIWM convention is to call **PopWindow** whenever a mousedown event occurs in the content region of a window.

```
DragWindow(theWindow)
Window     theWindow;
```

DragWindow allows the user to move a window with the cursor. While **DragWindow** is active, the cursor is displayed as a "move" icon. **DragWindow** retains "control" of the application program until the user releases the mouse button at the new location.

```
ReshapeWindow(theWindow)
Window    theWindow;
```

ReshapeWindow allows the user to resize a window with the cursor. While **ReshapeWindow** is active, the cursor is displayed as a "reshape" icon. **ReshapeWindow** retains "control" of the application program until the user releases the mouse button at the new location of the window's lower right corner.

```
SetWindow(theWindow)
Window    theWindow;
```

SetWindow designates a window as the "current graphics port"; i.e., as the target of all subsequent drawing commands until **SetWindow** is called again with a different window record.

```
Window    FindWindow(windowRegion)
int    *windowRegion;
```

FindWindow returns the window record of the window in which a mouse event occurred and updates windowRegion with the name of the region in that window in which the event occurred. Its uses in a multi-window interactive graphics program should be obvious.

```
Window    WhichWindow(gid)
int    gid;
```

WhichWindow returns the window record of the window with the graphic identifier (gid) specified by the argument. Its primary use is with the **REDRAW** event, which provides the gid of the window to be redrawn. See Section 2.5 (Programming Hints) of the IRIS User Guide Window Manager Appendix for more information.

```
DrawTitleBar(theWindow)
Window    theWindow;
```

DrawTitleBar draws the MIWM title bar in the top 16 pixels of a window.

Its use is optional, but if it is not called the user will not be able to see the drag region or the go-away box.

```
ResetWindowList()          /* repositions WindowList's pointer to
                              top-of-list */
int EndWindowList()        /* returns TRUE if no more window
                              records; else FALSE */
Window CurrentWindow()     /* returns the window record pointed to by
                              the list pointer */
GetNextWindow()            /* positions WindowList's pointer to the
                              next window record */
```

ResetWindowList, **EndWindowList**, **CurrentWindow** and **GetNextWindow** allow the programmer to obtain all the currently open windows operating under MIWM. Refer to the sample program *wintest.c* for examples of their use.

CAUTION: **CurrentWindow** returns an undefined value if called with EndWindowlist() == TRUE.

```
/* This is file wintest.c  It contains a main function to test
   the Modified IRIS Window Manager (MIWM)  */

#include "gl.h"
#include "device.h"
#include "windows.h"

/*********************************************
* Author:       Larry Griggs
* Written:      1 Apr 87
* Amended:      10 Jun 87
* FnName:       main
* Purpose:      test window manager
* Params:       argc, argv (unused)
* Returns:      void
* SideEffects:  none
*********************************************/

Object          testobj;   /* a global object which we will display in each window */

main(argc, argv)
int   argc;             /* unused */
char *argv[];           /* unused */
{
     short      data;           /* holds evt queue return value */
     short      active;         /* boolean: is this program active? */
     int   windowCode;  /* the region of the window: content,go-away,drag */
     char title[50];  /* title to be displayed on window title bar */
     Window  theWindow;   /* a window record */
     int   MainMenu;    /* for use with IRIS pop-up menu package */
     static int win_no=0;      /* counts number of windows that have been
                    opened since program start-up */

/*********************************************
                INITIALIZATION
*********************************************/


     sprintf(title,"Window %d",win_no++);       /* title to put in the title bar */

     /* open the first window as the background window,
```

```
  meaning that it can't be resized, moved, closed or popped */
prefposition(10,XMAXSCREEN-10,10,YMAXSCREEN);
OpenWindow(title,&testobj,TRUE); /* TRUE => background window */


TakeInputFocus(); /* ensure this program has "input focus"; i.e.
                     it is the designated recipient of all keyboard,
                     mouse and button-box events */
active = TRUE;          /* set boolean variable */

/* No IRIS graphics library functions can be called until
   after the first OpenWindow call. This is because this first call
   in turn calls various IRIS functions that perform necessary initiali-
   zations; having already called OpenWindow, we can now start using
   these library routines */

/* make the object.  In this case, just a blank screen */
testobj = genobj();
makeobj(testobj);
     color(CYAN);
     clear();
closeobj();

/* define the popup menu. */
DefineMenu(&MainMenu);

/* flush event queue */
qreset();

/* name devices to be queued */
qdevice(INPUTCHANGE);
qdevice(RIGHTMOUSE);
qdevice(LEFTMOUSE);
qdevice(REDRAW);

/* display the window we opened above */
ResetWindowList();          /* go to top of the windowlist */
theWindow = CurrentWindow(); /* put the window record that is at
                     the top of the windowlist into a
                     holding variable */
SetWindow(theWindow);          /* make this window the recipient of all
                     drawing commands */
callobj(*theWindow.obj);     /* call the Object whose address we
```

```
                    stored in the window record during the
                    call to OpenWindow */
    DrawTitleBar(theWindow);  /* draw the title bar */
    swapbuffers();            /* make the picture we've drawn visible */

/*****************************************
            START OF MAIN DISPLAY LOOP
*****************************************/

    while (TRUE)
    {
        /* check for any user-generated events */
        while (qtest())
        {
            switch (qread(&data))
            {
                case LEFTMOUSE:
                  if (data)    /* if a down event */
                  {
                    /* get the window and specific region in
                     which the mousedown event occurred */
                    theWindow = FindWindow(&windowCode);

                    switch(windowCode)
                    {
                        case IN_CONTENT:
                          PopWindow(theWindow);
                          break;

                        case IN_DRAG:
                          DragWindow(theWindow);
                          break;

                        case IN_GO_AWAY:
                          if (TrackGoAway(theWindow))
                              CloseWindow(theWindow);
                          break;

                        case IN_RESHAPE:
                          ReshapeWindow(theWindow);
                          break;

                        default:
```

```
            break;
        } /* end switch */
        break;

case RIGHTMOUSE:
        if (data)
        {
            sprintf(title,"Window %d",win_no++);
            DoMenu(dopup(MainMenu),title);
        }
        break;

case REDRAW:
        /* has a window been opened,
        moved, or resized?  If so,
        we only redraw the window actually
        involved */
        theWindow = WhichWindow((int) data);
        SetWindow(theWindow);
        reshapeviewport();
        frontbuffer(TRUE);
        callobj(*theWindow.obj);
        DrawTitleBar(theWindow);
        frontbuffer(FALSE);
        break;

case INPUTCHANGE:
    /* the input focus (ie active pgm) has changed */

    /* NON-PREFERRED METHOD */
    /* in an actual application, etiquette
       would require yielding input focus
       as shown below, but if this program
       was going to retain focus we would..
    TakeInputFocus();
    break;
    */

    /* PREFERRED METHOD */
    /* if this pgm is now active... */
    if (data)
       active = TRUE;
    else
```

```
            {
              /* otherwise save current windows
                 in both front and back buffers and
                 stand by to swapbuffers until this
                 program regains input focus */

              active = FALSE;
              frontbuffer(TRUE);
              ResetWindowList();
              while (!EndWindowList())
              {
                   theWindow = CurrentWindow();
                   SetWindow(theWindow);
                   callobj(*theWindow.obj);
                   DrawTitleBar(theWindow);
                   GetNextWindow();
              }
              frontbuffer(FALSE);
            }
            break;

            } /* end if (data) */
            break;

        default:
            break;
    } /* endswitch(qread()) */

} /* endwhile qtest()*/

/* swapbuffers until this program is reactivated */
if (!active)
    while (!qtest())
        swapbuffers();

/* draw the windows by going through the entire windowlist */
ResetWindowList();
while (!EndWindowList())
{
    theWindow = CurrentWindow();
    SetWindow(theWindow);
    callobj(*theWindow.obj);
    DrawTitleBar(theWindow);
```

```
            GetNextWindow();
        }

        swapbuffers();

/*****************************************
            END OF MAIN DISPLAY LOOP
*****************************************/

    } /* end while(TRUE) */
} /* end main */

/*****************************************
* FnTitle:      DefineMenu()
* Author:       Larry Griggs
* Date:         13 Dec 86
* Amended:      18 Feb 87
* Purpose:      Defines popup menus for main program (Refer to IRIS User's Guide)
* Returns:      altered MainMenu
* Side Effects: none
*****************************************/
DefineMenu(MainMenu)
int  *MainMenu;
{
    int  ControlMenu, HelpMenu;

    HelpMenu = defpup("Display Help %x111");
    ControlMenu = defpup("Quit %x9999");
    *MainMenu = defpup("Main Menu %t");
    addtopup(*MainMenu, "Help %x101 %m", HelpMenu);
    addtopup(*MainMenu, "Controls %x101 %m", ControlMenu);
    addtopup(*MainMenu, "Add Window %x999");
}

/*****************************************
* FnTitle:      DoMenu
* Author:       Larry Griggs
* Date:         13 Dec 86
* Amended:      10 Jun 87; 1 Apr 87; 26 Feb 87; 20 Feb 87
* Purpose:      Process user input to popup menus
* Params:       the menu hit number and the title of the window to open
* Returns:      void
* Side Effects: opens a new window if "Add Window" option selected
```

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
DoMenu(pupval,title)
int         pupval;
char        title[];
{
     switch(pupval)
     {
          case 111:
               /* HELP not implemented */
               break;
          case 9999:
               gexit();
               exit(0);
               break;
          case 999:
               /* unless some window constraint is specified now,
                  the last window constraint specified will be used.
                  If prefposition instead of minsize is called,
                  the window would be
                  immediately displayed without the user positioning
                  and sizing it */
               minsize(50,50);
               OpenWindow(title,&testobj,FALSE); /* FALSE => normal window */
               break;
          default:
               break;
     }
}
```

LIST OF REFERENCES

[ADAM87] Adams, Rodney M., "A Software Architecture for a Commander's Display System", Master's Thesis, Naval Postgraduate School, Monterey, California, March 1987.

[BERZ85] Berzins, Valdis and Michael Gray, "Analysis and Design in MSG.84: Formalizing Functional Specifications", *IEEE Transactions on Software Engineering* , Vol SE-11, No 8, August 1985, pp 657-670.

[BERZ86] Berzins, Valdis, et. al., "Abstraction-Based Software Development", *Communications of the ACM* , Vol 29, No 5, May 1986, pp 402-415.

[CASE86] Casey, Roger, "Advanced Technology Demonstration (ATD) Phase I Program Plan for Intra-Battle Group Networking (UNT/DCS)", Naval Ocean Systems Command, 4 December 1986.

[GRIG87] Griggs, L. W., "Memorandum for Record: Discussions between Mr. Roger Casey (NOSC UNT Project Manager) and L. W. Griggs (NPS thesis student) on 27-28 Jan 87", dtd 2 Feb 87.

[HORO83] Horowitz, Ellis and Sartaj Sahni, *Fundamentals of Data Structures,* Computer Science Press, Rockville, Maryland:1983.

[NAVSEA83] Advanced Combat Direction System (ACDS) Design Project, *ACDS Symbology Study: ACDS Block I Symbol Set,* Naval Sea Systems Command, Combat Direction System Division, NAVSEA 81Y, Washington D. C., November 1983.

[SPRA86] Spragins, John D. and Joseph L Hammond, "Multi-Network Controller Technology Assessment of Unified Networking Technology System Architecture", Final Technical Report to NOSC under Contract Number N660001-85-D-0203, San Diego State University, San Diego, CA, November 1986.

[STAL85] Stallings, William, *Data and Computer Communications* , Macmillan, New York:1985.

[TERP82] Terplan, K, "Network Performance Reporting", *ACM Computer Network Performance Symposium Performance Evaluation Review* , Vol 11, Number 1, Spring 1982, pp 156-170.

[ZYDA87] Zyda, Michael, "Research Proposal: High Performance Interactive Graphics for a Multinetwork Controller Monitoring Station (NPS Reference NC4(52ZK)/mjz)", dtd 6 Jan 87.

Distribution List for Dr. Michael J. Zyda

Defense Technical Information Center
Cameron Station
Alexandria, VA 22314 — 2 copies

Library, Code 0142
Naval Postgraduate School
Monterey, CA 93943 — 2 copies

Center for Naval Analyses
2000 N. Beauregard Street
Alexandria, VA 22311 — 1 copy

Director of Research Administration
Code 012
Naval Postgraduate School
Monterey, CA 93943 — 1 copy

Dr. Michael J. Zyda
Naval Postgraduate School
Code 52, Dept. of Computer Science
Monterey, California 93943-5100 — 150 copies

Mr. Russell Davis
HQ, USACDEC
Attention: ATEC-IM
Fort Ord, California 93941 — 1 copy

John Maynard
Naval Ocean Systems Center
Code 402
San Diego, California 92152 — 1 copy

El Wells
Naval Ocean Systems Center
Code 443
San Diego, California 92152 — 1 copy

Roger Casey
Naval Ocean Systems Center
Code 84
San Diego, California 92152 — 1 copy

Dr. Al Zied
Naval Ocean Systems Center
Code 433
San Diego, California 92152 — 1 copy